UNIVERSITY OF OSLO
Fysisk Institutt

Tracking of an Airplane using EKF and SPF

Master thesis

Rune Jansberg

May 31, 2010

Problem Description

Tracking of an Airplane using EKF and SPF.

The theory behind a Kalman filter requires that the system equations included in the Kalman filter algorithm are linear. A non-linear system must therefore be linearised - which gives the Extended Kalman filter (EKF). In recent years, non-linear filters, like the sigma point filters (SPF), have received increased attention. SPF is also called "Unscented Kalman Filter." This filter is based on statistical methods and requires more calculations than the Kalman filter, but in non-linear systems where the linearisation gives large errors, the SPF performs at least as good as the EKF.

In this thesis the filters will be compared on two systems; tracking of an object falling through the atmosphere, and tracking of an airplane. The performance of the different filters will be compared. Simulation is done in Matlab.

Suggestions for progress:

1. Familiarize yourself with the theory behind the Kalman filter, including the EKF and document the theory.

2. Familiarize yourself with the theory behind the SPF and document the theory.

3. Model and simulate in Matlab an object falling through the atmosphere, details are given by Kongsberg Defence Systems (KDS).

4. Estimate the position and velocity of the object using EKF and SPF and compare the results.

5. Make a model of a system tracking a plane with angle and distance measurements. To test the filter it is assumed that the aircraft has a sine perturbation in the horizontal plane, $50(m)\sin(2\pi t \times 0.1)$. The aircraft will have an altitude of 100m and a speed of 170 m/s. Use the EKF and SPF to estimate the position and speed of the plane with angle and distance measurements and compare the results. Assume that the perturbation is known/unknown to the filter.

6. We get a more challenging scenario by perturbing the trajectory in both the horizontal and vertical planes. Model perturbations of $50(m)\sin(2\pi t \times 0.1)$ in the horizon-

tal plane and $50(m)\cos(2\pi t \times 0.1)$ in the vertical plane. Do the same investigations as in section 5.

Preface

This is a master thesis for the undersigned student and the work is carried out at the Physical Institute of the University of Oslo in one semester, spring 2010. The field of study is Cybernetics located at the University Graduate Center at Kjeller.

The main task of this thesis is target tracking of an airplane with an oscillating motion with use of radar measurements and different non-linear filter approaches. This is done in cooperation with Kongsberg Defense Systems (KDS), and it is desirable that this thesis may be useful for future projects in target tracking.

I want to thank my teaching supervisor at UNIK Oddvar Hallingstad for advice and assistance of great value throughout the project. At KDS, I want to thank my supervisor Åge Skullestad for a challenging task and the help given both by telephone and visits at UNIK.

Kjeller, May 31, 2010

Rune Jansberg

Summary

The main purpose of this thesis is comparing two non-linear filters in target tracking using radar measurements. These two filters are Extended Kalman filter (EKF) and Sigma Point filter (SPF). This task was given by Kongsberg Defence Systems (KDS). Kongsberg Defense Systems use radars and recursive filters to track airplanes.

Sigma Point filter (SPF), also known as the Unscented Kalman filter (UKF), and the Extended Kalman filter have been compared to each other, looking at performance of estimating states and parameters from radar measurements in spherical coordinates. Airplane trajectories with perturbations in the horizontal plane and pertubations in both horizontal and vertical plane as sines and cosines respectively have been estimated. Estimating trajectories are often related to, including estimating the states, to estimate system parameters as well. The filters have been compared using both unknown and known process models. Three scenarios have been modelled where the airplane is heading towards the radar from east, from north east and north against east away from the radar. In addition an object falling through the atmosphere straight above the radar have been looked into. Matlab have been used for the simulations.

A theoretical evaluation of EKF and SPF are presented indicating that SPF will performe better than EKF. This is later confirmed with simulations. SPF gives better estimates than EKF estimating the states of the airplane with both linear and non-linear process filter models. Parameters related to the airplane motion are heading angle and angular velocity. The angular velocity are estimated using Multiple Signal Classification (MUSIC). This was done mainly because it was found that the filters had problems estimating the frequancy needed to compute the angular velocity. Heading angle are estimated by use of a relation in the filters measurement equation giving information needed for the filters. SPF is concluded being the best choice in target tracking of airplanes simulated in this thesis. The same conclusions have been made for the falling body scenario, where SPF performed slightly better than EKF.

Table of Contents

1 Introduction **1**
- 1.1 Motivation and Background . . . 1
- 1.2 Introduction to Estimation . . . 2
- 1.3 Introduction to Tracking . . . 2
- 1.4 Thesis Outline . . . 3

2 Mathematical Background **5**
- 2.1 Stochastic Variables . . . 5
- 2.2 Stochastic Processes . . . 7
- 2.3 Non-Linear Systems and Linearization . . . 8
- 2.4 Rotation Matrices . . . 9
- 2.5 Euler Methods . . . 10
- 2.6 Unbiased Converted Measurements for Tracking . . . 11
  - 2.6.1 2D Measurements . . . 11
  - 2.6.2 3D Measurements . . . 12

3 Linear Filtering **15**
- 3.1 The Discrete-Time Linear Kalman Filter . . . 15
  - 3.1.1 Implementation of the Linear Kalman Filter . . . 17

4 Non-Linear Filtering **19**
- 4.1 Extended Kalman Filter . . . 19
  - 4.1.1 The discrete-time extended Kalman filter . . . 19
  - 4.1.2 Implementation of the Extended Kalman Filter . . . 20
  - 4.1.3 EKF and its Flaws . . . 21
- 4.2 Sigma Point Filter . . . 22
  - 4.2.1 Means and covariances of non-linear transformations . . . 22
  - 4.2.2 Unscented transformations . . . 25
  - 4.2.3 The Sigma point filter . . . 27

5 Parameter Estimation **33**
- 5.1 Kalman Filter Parameter Estimation . . . 33

5.2 Multiple Signal Classification (MUSIC) 34

6 Mathematical Modelling **37**
6.1 Horizontal Wave Motion 37
6.2 Helix Motion 42
6.3 The Filter Models 43
6.3.1 EKF with linear process model 44
6.3.2 SPF with linear process model 45
6.3.3 EKF with non-linear process model 46
6.3.4 SPF with non-linear process model 48
6.3.5 Summary 49
6.4 Description of Solution 50
6.5 Falling Body 55
6.5.1 Modelling of a falling body 55
6.5.2 The Filter Models 55

7 Results **57**
7.1 Horizontal wave motion 58
7.1.1 Linear process filter model with non-linear measurement equation . 58
7.1.2 Non-linear process filter model 61
7.2 Helix motion 71
7.2.1 Linear process filter model with non-linear measurements 71
7.2.2 Non-linear process filter model 73
7.3 Falling body 82

8 Conclusion **85**
8.1 Discussion 85
8.2 Conclusion 86
8.3 Recommendations 87

A Derivation of the Kalman Filter Equations **89**

B Simulation Results for Linear Process Filter Models **93**
B.1 Horizontal Wave Motion 93
B.2 Helix Motion 95

C Program Code **103**

Bibliography **156**

List of Figures

2.1 Zero mean white noise. 8
2.2 The relation $\vec{r} = \vec{r}_{qp} + \vec{\rho}$ 10

4.1 Linearized and nonlinear mean of 300 randomly generated points. $\tilde{r}$ uniformly distributed between ± 0.01 and $\tilde{\Theta}$ uniformly distributed between ± 0.35 radians. 23
4.2 Linearized and nonlinear mean and covariance of 300 randomly generated points. $\tilde{r}$ uniformly distributed between ± 0.01 and $\tilde{\Theta}$ uniformly distributed between ± 0.35 radians. 26
4.3 Linearized, nonlinear and UT mean and covariance of 300 randomly generated points. $\tilde{r}$ uniformly distributed between ± 0.01 and $\tilde{\Theta}$ uniformly distributed between ± 0.35 radians. 27

6.1 Radar frame, $\mathcal{F}_r$, and airplane frame, $\mathcal{F}_a$. $\underline{\mathrm{p}}^r_{ra}$ is the vector from origo in $\mathcal{F}_r$ to $\mathcal{F}_a$. ψ is the azimuth angle and υ the elevation angle. Φ is the heading angle of the airplane, rotated around the z axis of $\mathcal{F}_a$, giving the relationship: $\underline{\mathrm{p}}^r = \underline{\mathrm{p}}^r_{ra} + R^r_a(\Phi)\underline{\mathrm{p}}^a$. 38
6.2 Horizontal wave motion of an airplane seen from $\mathcal{F}_a$ with amplitude of 50 meters and angular velocity of $\omega = 2\pi \times 0.1$. 40
6.3 Horizontal wave motion of an airplane seen from $\mathcal{F}_r$ with amplitude of 50 meters and angular velocity of $\omega = 2\pi \times 0.1$. Heading straight towards the radar. 41
6.4 Helix motion of an airplane seen from $\mathcal{F}_a$ with amplitude of 50 meters and angular velocity of $\omega = 2\pi \times 0.1$. 43
6.5 Helix motion of an airplane seen from $\mathcal{F}_r$ with amplitude of 50 meters and angular velocity of $\omega = 2\pi \times 0.1$. Heading straight towards the radar. . . . 44
6.6 Radar frame, $\mathcal{F}_r$, and airplane frame, $\mathcal{F}_a$. Course along the x axis of $\mathcal{F}_a$ which is unknown for the filters. This is treated with linear regreassin of the measurements which gives an approximation of the course in green. . . 53
6.7 Falling body seen from the radar. Simulated for 30 seconds. 56

7.1 Scenario 1. Airplane position and velocities seen from $\mathcal{F}_r$ heading straight towards the radar from east. Perturbations unknown for the filters. 59

7.2 Scenario 1. Airplane position errors seen from $\mathcal{F}_r$ heading straight towards the radar from east. Perturbations unknown for the filters. 60
7.3 Scenario 1. Airplane velocity errors seen from $\mathcal{F}_r$ heading straight towards the radar from east. Perturbations unknown for the filters. 60
7.4 Scenario 1. Airplane position and velocity seen from $\mathcal{F}_r$. Perturbations known by the filters. 61
7.5 Scenario 1. Airplane position errors seen from $\mathcal{F}_r$. Perturbations known by the filters. 62
7.6 Scenario 1. Airplane velocity errors seen from $\mathcal{F}_r$. Perturbations known by the filters. 63
7.7 Scenario 1. Estimated $\underline{\mathrm{p}}^r_{ra}$ errors seen from $\mathcal{F}_r$. Perturbations known by the filters. 63
7.8 Scenario 1. Estimated ω and Φ errors. Perturbations known by the filters. 64
7.9 Scenario 1. Standard deviations from filters and true standard deviations. . 64
7.10 Scenario 2. Airplane position and velocity seen from $\mathcal{F}_r$. Perturbations known by the filters. 65
7.11 Scenario 2. Airplane position errors seen from $\mathcal{F}_r$. Perturbations known by the filters. 66
7.12 Scenario 2. Airplane velocity errors seen from $\mathcal{F}_r$. Perturbations known by the filters. 66
7.13 Scenario 2. Estimated $\underline{\mathrm{p}}^r_{ra}$ errors seen from $\mathcal{F}_r$. Perturbations known by the filters. 67
7.14 Scenario 2. Estimated ω and Φ errors. Perturbations known by the filters. 67
7.15 Scenario 3. Airplane position and velocity seen from $\mathcal{F}_r$. Perturbations known by the filters. 68
7.16 Scenario 3. Airplane position errors seen from $\mathcal{F}_r$. Perturbations known by the filters. 69
7.17 Scenario 3. Airplane velocity errors seen from $\mathcal{F}_r$. Perturbations known by the filters. 69
7.18 Scenario 3. Estimated $\underline{\mathrm{p}}^r_{ra}$ errors seen from $\mathcal{F}_r$. Perturbations known by the filters. 70
7.19 Scenario 2. Estimated ω and Φ errors. Perturbations known by the filters. 70
7.20 Scenario 1. Airplane position and velocities seen from $\mathcal{F}_r$ heading towards the radar east. Perturbations unknown for the filters. 71
7.21 Scenario 1. Airplane position errors seen from $\mathcal{F}_r$ heading towards the radar east. Perturbations unknown for the filters. 72
7.22 Scenario 1. Airplane velocity errors seen from $\mathcal{F}_r$ heading towards the radar east. Perturbations unknown for the filters. 72
7.23 Scenario 1. Airplane position and velocity seen from $\mathcal{F}_r$. Perturbations known by the filters. 73
7.24 Scenario 1. Airplane position errors seen from $\mathcal{F}_r$. Perturbations known by the filters. 74
7.25 Scenario 1. Airplane velocity errors seen from $\mathcal{F}_r$. Perturbations known by the filters. 74

7.26 Scenario 1. Estimated $\underline{p}^r_{ra}$ errors seen from $\mathcal{F}_r$. Perturbations known by the filters. 75
7.27 Scenario 1. Estimated ω and Φ errors. Perturbations known by the filters. 75
7.28 Scenario 1. Standard deviations from filters and true standard deviations. . 76
7.29 Scenario 2. Airplane position and velocity seen from $\mathcal{F}_r$. Perturbations known by the filters. 77
7.30 Scenario 2. Airplane position errors seen from $\mathcal{F}_r$. Perturbations known by the filters. 77
7.31 Scenario 2. Airplane velocity errors seen from $\mathcal{F}_r$. Perturbations known by the filters. 78
7.32 Scenario 2. Estimated $\underline{p}^r_{ra}$ errors seen from $\mathcal{F}_r$. Perturbations known by the filters. 78
7.33 Scenario 2. Estimated ω and Φ errors. Perturbations known by the filters. 79
7.34 Scenario 3. Airplane position and velocity seen from $\mathcal{F}_r$. Perturbations known by the filters. 80
7.35 Scenario 3. Airplane position errors seen from $\mathcal{F}_r$. Perturbations known by the filters. 80
7.36 Scenario 3. Airplane velocity errors seen from $\mathcal{F}_r$. Perturbations known by the filters. 81
7.37 Scenario 3. Estimated $\underline{p}^r_{ra}$ errors seen from $\mathcal{F}_r$. Perturbations known by the filters. 81
7.38 Scenario 2. Estimated ω and Φ errors. Perturbations known by the filters. 82
7.39 Falling body position and velocity seen from radar. 83
7.40 Falling body position errors, velocity errors and ballistic constant errors. . 83

B.1 Scenario 2, wave motion. Airplane position and velocities seen from $\mathcal{F}_r$. Perturbations unknown for the filters. 94
B.2 Scenario 2, wave motion. Airplane position errors seen from $\mathcal{F}_r$. Perturbations unknown for the filters. 94
B.3 Scenario 2, wave motion. Airplane velocity errors seen from $\mathcal{F}_r$. Perturbations unknown for the filters. 95
B.4 Scenario 3, wave motion. Airplane position and velocities seen from $\mathcal{F}_r$. Perturbations unknown for the filters. 96
B.5 Scenario 3, wave motion. Airplane position errors seen from $\mathcal{F}_r$. Perturbations unknown for the filters. 96
B.6 Scenario 3, wave motion. Airplane velocity errors seen from $\mathcal{F}_r$. Perturbations unknown for the filters. 97
B.7 Scenario 2, helix motion. Airplane position and velocities seen from $\mathcal{F}_r$. Perturbations unknown for the filters. 98
B.8 Scenario 2, helix motion. Airplane position errors seen from $\mathcal{F}_r$. Perturbations unknown for the filters. 98
B.9 Scenario 2, helix motion. Airplane velocity errors seen from $\mathcal{F}_r$. Perturbations unknown for the filters. 99

B.10 Scenario 3, helix motion. Airplane position and velocities seen from $\mathcal{F}_r$. Perturbations unknown for the filters. 100
B.11 Scenario 3. Airplane position errors seen from $\mathcal{F}_r$, helix motion. Perturbations unknown for the filters. 100
B.12 Scenario 3, helix motion. Airplane velocity errors seen from $\mathcal{F}_r$. Perturbations unknown for the filters. 101

List of Algorithms

1 Linear Kalman filter 17
2 Extended Kalman filter 20
3 Sigma Point Filter 32
4 Program Flow 54

Notation

Nomenclature	
Notation	Definition
x	Scalar
$\underline{x}$	Vector
A	Matrix
$\underline{x}_k$	Vector at discrete time t_k
$\underline{\bar{x}}$	Predicted state, time update
$\underline{\hat{x}}$	Estimated state, measurement update
$\dot{x}$	Differentaiation with respect of time
$\mathcal{F}_r$	r-frame
$\underline{p}^r$	Position represented in r-frame
$\underline{v}^r$	Velocity represented in r-frame
$\underline{p}^r_{ra}$	Vector between origin in $\mathcal{F}_r$ to $\mathcal{F}_a$ represented in $\mathcal{F}_r$
r	Range
ψ	Azimuth angle
υ	Elevation angle
ϕ	Heading angle
$Pr(E)$	Probability of event E
$f_X(x) = \frac{\partial F_x(x)}{\partial x}$	Probability density function of X
$F_X(x) = P(X \leq x)$	Probability distribution of X
$\underline{\bar{x}} = E[\underline{x}]$	Mean of $\underline{x}$
σ_x	Standard deviation of x
Φ_{xx}	Autocorrelation
ϕ_{xy}	Crosscorrelation
$\Phi_{xx}(\omega)$	Power spectral density

Acronyms	
Acronym	Definition
EKF	Extended Kalman filter
pdf	Probability density function
PDF	Probability distribution function
SPF	Sigma point filter

Chapter 1

Introduction

1.1 Motivation and Background

Kongsberg Defence Systems are Norway's premier supplier of defence and aerospace-related systems, and they use Kalman filters in many of their products. An important aspect of these systems are radars that monitor the airspace and tracking of airplanes. Here the accuracy of the target tracking is essential. In many years the Extended Kalman filter have played the main role of non-linear estimation. In the later years the Sigma Point filter have received increased attention. This constitutes the motivation of this work, to study the accuracy of target tracking with these filters, and compare to see if one performes any better than the other.

It will be looked into tracking of an object falling through the atmosphere. This is a problem studied in many scenarios the last decades. Three books dealing with this can be seen in the bibliography of this thesis [1, 4, 2]. [2] gives a long list of references looking at tracking of ballistic objects, i.e. falling bodies. [4, 2] gives a comparison of EKF and SPF estimating position and velocity of the falling body and concludes that SPF gives better results than EKF.

In [12] EKF and SPF are compared in target tracking in mixed coordinates. Mixed coordinates means that the measurements received from the radar are in spherical coordinates, and that the estimation happens in cartesian coordinates. This gives a non-linear measurement equation with transformation from cartesian to spherical coordinates. However, here they compares the filters on eight scenarios between north and east with linear process filter model. The conclusions given are that SPF performes better than EKF in all of the scenarios.

This will also be looked into in this thesis. Tracking of an airplane using EKF and SPF with comparisons. Three scenarios are chosen to be modelled. The scenarios are against the radar from east, towards the radar from north east, and away from the radar from north against east. This will be studied for a horizontal wave motion, and a helix motion using mixed coordinates, where the airplane motion are known and unknown for the filters. This indicates that, when the motions are unknown it will be used a linear process filter model as in [12]. In addition to this it will be used a non-linear process filter model where the motions are known for the filters. As mentioned, the filters will be

compared and see if they gives the same results as mentioned above for both known and unknown airplane motion.

1.2 Introduction to Estimation

In the later years, estimation are used in many applications of determening position and velocity of aircrafts, dynamic positioning of marine vessels, determination of model parameters for predicting the state of a physical system, in control theory, in economics and so on. Bar-Shalom, Li and Kirubarajan defines estimation as a "process of inferring the value of quantity of interest from indirect, inaccurate and uncertain observations" [22]. Probably the first use of estimation was by Laplace, Legendre and Gauss as they wanted to determine planet orbit parameters by least-squares techniques [22].

In order to estimate a system, minimum two models are required. The process model, the model which describes the evolution of the state with time; the dynamic model. That is a differential equation, or a set of differential equations. The second model is the measurement model. This is an algebraic equation with noise. These two models gives us the system of process model and measurement model.

An optimal estimator is an algorithm that use measurements and calculates a minimum variance estimate of a variable with use of the knowledge about the process and the measurement equation with its statistics [22]. This variable to be estimated, can be a time-invariant parameter like a scalar, a vector or a matrix. It may also be the state of a dynamic system that evolves over time with stochastic disturbances. These are the two main classes of estimators, the parameter estimators, and the state estimators. Bar-Shalom, Li and Kirubarajan in [22] states that the optimal estimator have some advantages and disadvantages. The advantages is of course that it makes the best utilization of the knowledge of the system and the noise. The disadvantages are that it may be sensitive to modelling errors and computationally expensive. The linear Kalman filter is an optimal estimation algorithm [1, 2, 4, 22].

1.3 Introduction to Tracking

Target tracking are used in many important applications of surveillance, guidence and obstacle avoidance systems. Where the main tasks are to determine position, velocity, acceleration and course of one or more moving targets. In a tracking system like this, some types of sensors are needed to give measurements of and/or from the targets. This depends on if the target is a known friendly passenger aircraft giving measurement information from the aircraft to ground airport, or a hostile aircraft that the only measurements available are from your own surveillance systems at fixed locations or on moving platforms. The sensors giving measurements can be radars, cameras, infrared cameras, sonars etc. The measurements will have some kind of measurement noise depending on the sensor, one sensor may be inaccurate on far range, an other more inaccurate on short range giving good measurements on far range.

To handle a complex tracking problem, some kind of a recursive filter for target state

estimation is needed. The term ”filter” are used because one needs to ”filter” out the noise, or eleminate an undesired signal, from the measurements. The Kalman filter algorithm, developed for about 50 years ago [15], is a well known filter used on linear Gaussian problems. In the later years nonlinear filtering have become more and more in the focus of interest. Since the Kalman filter requires linear system equations, nonlinear systems must be linearised, which leads to the Extended Kalman filter (EKF). An other nonlinear approach is the Sigma Point filter (SPF), also known as Unscented Kalman filter. These types of filters in the context of target tracking will be handled in this thesis.

In target tracking the best way to describe the target motion is usually in cartesian coordinates. The measurements from a sensor like a radar, are usually in spherical coordinates. This leads to a non-linear measurement model with the transformation from cartesian to spherical coordinates.

1.4 Thesis Outline

This thesis chapters will here be briefly discribed. In addition to the introduction in this chapter, the thesis is structured as follows:

- Chapter 2: Mathematical background. There are presented an introduction to different mathematical theories containing stochastic variables and processes, non-linear and linear systems with linearizations, rotation matrices etc.
- Chapter 3: Linear filtering. Theory of the linear Kalman filter is presented.
- Chapter 4: Non-linear filtering. Theory of the Extend Kalman filter and the Sigma point filter are presented.
- Chapter 5: Parameter estimation. Theory of parameter estimation using Kalman filters, and theory of frequency estimation using MUSIC.
- Chapter 6: Mathematical modelling. The simulation models and filter models are derived.
- Chapter 7: Results. Main results are presented with comments and compared with related work.
- Chapter 8: Discussion, conclusion and recommendations.

Chapter 2

Mathematical Background

This mathematical background can be read in more detail in [1, 3, 4, 9, 13, 14, 16].

2.1 Stochastic Variables

The theory which examine stochastic variables and processes are taken from [1, 16]. A stochastic variable, or a random variable, is often referd to as X. It is a variable that will have a completely random value and are expected to change value every time it is inspected. It can be defined for both continous and discrete time as a function of the outcomes of some random experiment. If $Pr(E)$ is the probability of a possible event E, the probability where different values are taken by a random variable is specified by the probability distribution function (PDF) $F(x)$, which is defined by

$$F_X(x) = Pr(X \leq x) \tag{2.1}$$

or by the probability density function(pdf) $f(x)$:

$$f_X(x) = \frac{dF_X(x)}{dx} \tag{2.2}$$

The PDF is a function that gives the probability of a random value $X \in \mathcal{R}$ will take a value less than or equal to some value x. Where the pdf is the derivative of the PDF. The pdf have som properties as:

$$f_X(x) \leq 0, \text{ for all x } \in \mathcal{R} \tag{2.3}$$

$$\int_{-\infty}^{\infty} f_X(x)dx = 1 \tag{2.4}$$

$$Pr(a < X < b) = \int_a^b f_X(x)dx \tag{2.5}$$

And the properties of the PDF are:

$$F_X(x) \in [0,1] \tag{2.6}$$
$$F_X(-\infty) = 0 \tag{2.7}$$
$$F_X(\infty) = 1 \tag{2.8}$$
$$F_X(a) \leq F(b), \text{ if } a \leq b \tag{2.9}$$
$$Pr(a \leq X \leq b) = F_X(b) - F_X(a) \tag{2.10}$$

The expectation of a random variable is defined as:

$$\bar{x} = E[X] = \int_{-\infty}^{\infty} x f_X(x) dx \tag{2.11}$$

which is the sum of all values the random variable may take and weighted by the probability the value may take. This is also called the mean of X or the first moment of X. This mean is the average value of infinitely number of samples.

The variance of a random number are defined as:

$$Var(X) = \sigma^2 \tag{2.12}$$
$$\sigma^2 = \int_{-\infty}^{\infty} (x - E[X])^2 f_X(x) dx \tag{2.13}$$
$$= E[(X - \bar{x})^2] \tag{2.14}$$

where σ is called the standard deviation of the random variable. The variance describes the spread of the samples around the mean.

The covariance is a degree of how much a random variable is related to another and is defined as:

$$Cov(X,Y) = E[(X - \bar{x})(Y - \bar{y})] \tag{2.15}$$

The covariance of a vector $\underline{x}$ with n elements, can be defined as:

$$P = E[(\underline{X} - \bar{\underline{x}})(\underline{X} - \bar{\underline{x}})^T] \tag{2.16}$$
$$= \int_{-\infty}^{\infty} (\underline{x} - \bar{\underline{x}})((\underline{x} - \bar{\underline{x}}) f(\underline{x}) \tag{2.17}$$

where P is a matrix containing $n \times n$ values. On the diagonal of P, we can find the variance of $x_1, x_2, \ldots, x_n$.

The most common probability distribution is the normal distribution, also known as the gaussion distribution, where the pdf is defined as:

$$f_X(\underline{x}) = \frac{1}{(2\pi)^{\frac{n}{2}} |P|^{\frac{1}{2}}} e^{-\frac{1}{2}(\underline{x} - \bar{\underline{x}})^T P^{-1} (\underline{x} - \bar{\underline{x}})} \tag{2.18}$$

The area under this curve is unity. A common notation for the normal distribution of a random variable is:

$$\underline{X} \sim N(\bar{\underline{x}}, P) \tag{2.19}$$

where $\underline{X}$ is normal distributed around the mean $\underline{\bar{x}}$ with standard deviation $\sqrt{P}$.

Another common distribution is the uniform distribution, which is characterized by a constant probability density, over a finite range. If we have a random variable on the range from a to b, the pdf is given by:

$$f_X(x) = \frac{1}{b-a} \tag{2.20}$$

2.2 Stochastic Processes

A stochastic process is a collection, or ensamble, of functions of time that may be observed on any inspection of an experiment. The ensamble may be any finite number, a countable infinity, or a noncountable infinity of functions. If an random variable is input to a function, the output will be a transformed random variable.

The statistical relation of the samples of a stochastic process at two different time moments are the autocorrelation defined by:

$$\phi_{xx} = E[x(t_1)x(t_1+\tau)] \tag{2.21}$$

$$\phi_{xx}(t_1,t_2) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} x(t_1)x(t_1+\tau)f_X(x(t_1),x(t_2))dx(t_1)dx(t_2) \tag{2.22}$$

and the crosscorrelation is defined as:

$$\phi_{xy} = E[x(t_1)y(t_2)] \tag{2.23}$$

$$\phi_{xy}(t_1,t_2) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} x(t_1)y(t_1+\tau)f_X(x(t_1),y(t_2))dx(t_1)dy(t_2) \tag{2.24}$$

If $E[x(t_1)]$, $E[x(t_2)]$ and $E[y(t_2)]$ are zero, these correlations becomes the covariance of the random variables. If a stochastic process is stationary, the statistical properties are invariant in time. If $t_2 = t_1 + \tau$, the correlation functions are functions only of τ:

$$\phi_{xx}(t_1,t_2) = \phi_{xx}(\tau) = E[x(t_1)x(t_1+\tau)] \tag{2.25}$$

$$\phi_{xy}(t_1,t_2) = \phi_{xy}(\tau) = E[x(t_1)y(t_1+\tau)] \tag{2.26}$$

In association to a stationary stochastic process, the power spectral density (PSD) is a function of a frequency variable instead of time. Which defines how the power of a signal is distributed with frequency. One may look at the PSD as a indication of at wich frequencies variations are strong, and at which frequencies variations are weak. The power of a stochastic process, can be defined as the expected squared value of the members of the ensamble. Given the autocorrelation $\Phi_{xx}(\tau)$, then the power spectral density is defined as:

$$\Phi_{xx}(\omega) = \int_{-\infty}^{\infty} \Phi_{xx}(\tau)e^{-j\omega\tau}d\tau \tag{2.27}$$

$$\phi_{xx}(\tau) = \frac{1}{2\pi}\int_{-\infty}^{\infty} \Phi_{xx}(\omega)e^{j\omega\tau}d\omega \tag{2.28}$$

White noise

White noise is a random signal where the PSD is constant as $\Phi_{xx}(\omega) = \Phi_0$. This indicates a flat PSD, and that power is distributed uniformly over all frequency components in a full infinite range. This infinite-bandwith white noise exists only in theory, because to have power at all frequencies will give a total power of a signal to be infinite, and impossible to generate. A theoretical example of a Gaussian white noice process can be seen in figure 2.1

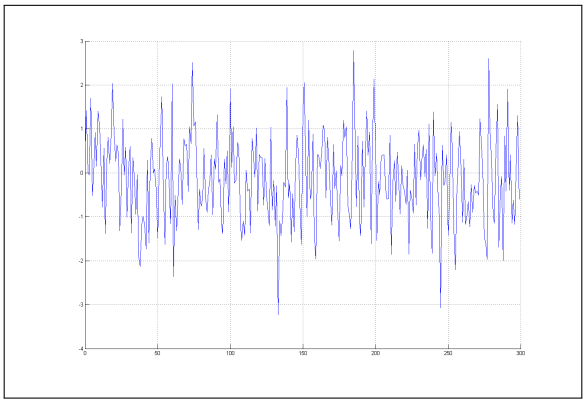

Figure 2.1: Zero mean white noise.

2.3 Non-Linear Systems and Linearization

This section dealing with non-linear systems and linearization, are taken from [4, 3]. Linear systems do not exist in the real world [4], even the simplest systems are operating non-linearly over some time periods. A non-linear continous system can be written as:

$$\dot{\underline{x}} = \underline{f}(\underline{x}, \underline{u}, \underline{v}) \tag{2.29}$$

$$\underline{z} = \underline{h}(\underline{x}, \underline{w}) \tag{2.30}$$

where $\underline{f}$ and $\underline{h}$ are vector functions, $\underline{v}$ and $\underline{w}$ are som type of process noise and measurement noise respectively. The system is linear if it is on the form:

$$\underline{f}(\underline{x}, \underline{u}, \underline{v}) = F\underline{x} + L\underline{u} + G\underline{v} \tag{2.31}$$

$$\underline{h}(\underline{x}, \underline{w}) = H\underline{x} + \underline{w} \tag{2.32}$$

This formulation can be useful in many applications, and the linear system theory may often be applicable. To use this linear system theory on non-linear systems, the system must be linearized to fit the non-linear system over some time periods. If we have the non-linear vector function $\underline{f}(\underline{x})$ where $\underline{x}$ is a $n \times 1$ vector, then the linearization is given by a Taylor series expansion of $\underline{f}$ around an operating point $\underline{x} = \bar{\underline{x}}$, where we can define $\tilde{\underline{x}} = \underline{x} - \bar{\underline{x}}$:

$$\underline{f}(\underline{x}) = \underline{f}(\bar{\underline{x}}) + D_{\tilde{\underline{x}}}\underline{f} + \frac{1}{2!}D^2_{\tilde{\underline{x}}}\underline{f} + \frac{1}{2!}D^3_{\tilde{\underline{x}}}\underline{f} + \dots \tag{2.33}$$

where

$$D^k_{\underline{\tilde{x}}}\underline{f} = \Big(\sum_{i=1}^{n} \tilde{x}_i \frac{\partial}{\partial x_i}\Big)^k \underline{f}(\underline{x}) \mid_{\underline{\bar{x}}} \tag{2.34}$$

$\frac{\partial \underline{f}}{\partial \underline{x}}$ is called the Jacobian of $\underline{f}$. For a 3×1 matrix, it is given as:

$$F = \frac{\partial \underline{f}}{\partial \underline{x}} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \frac{\partial f_1}{\partial x_3} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \frac{\partial f_2}{\partial x_3} \\ \frac{\partial f_3}{\partial x_1} & \frac{\partial f_3}{\partial x_2} & \frac{\partial f_3}{\partial x_3} \end{bmatrix} \Bigg|_{\underline{\bar{x}}} \tag{2.35}$$

Here the the name of the Jaboian is F to easy see the relation from equation 2.31

$$\dot{\underline{\tilde{x}}} = F\underline{\bar{x}} \tag{2.36}$$

2.4 Rotation Matrices

The theory of rotation matrices are taken from [9]. In many applications of aerospace, marine systems, robotics and navigation one may need to represent vectors in different cartesian coordinate systems. If we have two coordinate frames $\mathcal{F}_q$ and $\mathcal{F}_p$, we can represent a vector $\underline{\rho}^p$ in $\mathcal{F}_q$ with a rotation matrix. The superscript in $\underline{\rho}^p$ indicates which of the coordinate frames the vector is represented in.

We define the frames with orthogonal unit vectors in an affine space as:

$$\mathcal{F}_q = \{O_q; \vec{q}_1, \vec{q}_2, \vec{q}_3\} \tag{2.37}$$

$$\mathcal{F}_p = \{O_p; \vec{p}_1, \vec{p}_2, \vec{p}_3\} \tag{2.38}$$

where O_q and O_p are the origin i $\mathcal{F}_q$ and $\mathcal{F}_p$ respectively. If we have a point P in $\mathcal{F}_q$ that are represented in frame $\mathcal{F}_p$, the point P can then be described by two geometric vectors $\vec{r}$ and $\vec{\rho}$ as:

$$P = O_q + \vec{r} = O_p + \vec{\rho} \tag{2.39}$$

The substraction of two points in an affine space is a vector giving the relation:

$$\vec{r}_{qp} = O_p - O_q \tag{2.40}$$

$$\vec{r} = \vec{r}_{qp} + \vec{\rho} \tag{2.41}$$

This relations are shown in figure 2.2.

These geometric vectors can be represented in the different orthonormal frames by algebraic position vectors $\underline{r}^q$ and $\underline{\rho}^p$. If we represent the vector between the origin in $\mathcal{F}_q$ to $\mathcal{F}_p$ in $\mathcal{F}_q$ as $\underline{r}^q_{qp}$ we have the equation:

$$\underline{r}^q = \underline{r}^q_{qp} + R^q_p(\Phi)\underline{\rho}^p \tag{2.42}$$

Here Φ denotes the angle which the frame $\mathcal{F}_p$ is rotated counter clockwise from $\mathcal{F}_q$. In figure 2.2 $\mathcal{F}_p$ is rotated 120 degrees seen from $\mathcal{F}_q$ around $\vec{p}_3$ (z axis). The rotation matrices for rotations around the three axes are defined as:

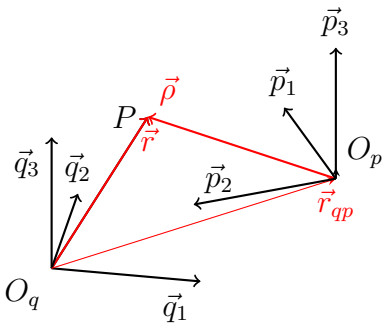

Figure 2.2: The relation $\vec{r} = \vec{r}_{qp} + \vec{\rho}$

$$R_1 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\Phi & -\sin\Phi \\ 0 & \sin\Phi & \cos\Phi \end{bmatrix}, R_2 = \begin{bmatrix} \cos\Phi & 0 & \sin\Phi \\ 0 & 1 & 0 \\ -\sin\Phi & 0 & \cos\Phi \end{bmatrix}, R_3 = \begin{bmatrix} \cos\Phi & -\sin\Phi & 0 \\ \sin\Phi & \cos\Phi & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (2.43)$$

The rotation matrices are orthogonal which implies that $R_p^q = (R_q^p)^{-1} = (R_q^p)^T$.

2.5 Euler Methods

To simulate a system of continous differential equations a numerical integration scheme is needed. A simple and important integration scheme is Euler's method where the numerical solution is computed as in equation 2.44 [14]:

$$\underline{y}_{k+1} = \underline{y}_k + T\underline{f}(\underline{y}_k, t_k) \quad (2.44)$$

where T is the time-step. This method is of order one and can give unsatisfactory results for non-linear systems, especially if T is to large.

To give a solution of higher order, the improved Euler method can be used. This is a 2. order integration scheme [14]. Here, Euler's method includes an evaluation $\bar{\underline{y}}_{k+1} = \underline{y}_k + T\underline{f}(\underline{y}_k, t_k)$ where an approximation of $\underline{f}(\bar{\underline{y}}_{k+1}, t_{k+1})$ at time t_{k+1} is computed using $\bar{\underline{y}}_{k+1}$. This is used to improve the accuracy of the numerical solution $\underline{y}_{k+1}$. The method can be computed as:

$$\underline{k}_1 = \underline{f}(\underline{y}_k, t_k) \quad (2.45)$$

$$\underline{k}_2 = \underline{f}(\underline{y}_k + T\underline{k}_1, t_k + T) \quad (2.46)$$

$$\underline{y}_{k+1} = \underline{y}_k + \frac{T}{2}(\underline{k}_1 + \underline{k}_2) \quad (2.47)$$

There may be numerical integration errors if the system has dynamics that are faster than the discretization time T. The system can then be unstable having divergence problems, and the need of a higher order numerical integration scheme is needed, e.g. Runge Kutta methods are often used. But for the simulations in this thesis, the improved Euler method is found satisfactory.

2.6 Unbiased Converted Measurements for Tracking

Longbin, Xiaoquan, Zhou Yiyu, Sun Zhong Kang and Bar-Shalom, presented a paper in 1998 [13] about unbiased converted measurements for tracking. The unbiased polar to cartesian conversion from this paper are presented here in the following section.

2.6.1 2D Measurements

Consider the measurements vector

$$\underline{z} = \begin{bmatrix} r \\ \psi \end{bmatrix} \tag{2.48}$$

where r is range, and ψ is angles. The classical converted measurements from spherical to cartesian coordinates are

$$x_m = r_m \cos \psi_m \tag{2.49}$$
$$y_m = r_m \sin \psi_m \tag{2.50}$$

This transformation can give biased estimates because of the nonlinear transformation of the noisy angle measurement, that is w_ψ assumed to have a symmetric probability density function(pdf). This gives

$$E[\sin w_\psi] = 0 \tag{2.51}$$

Taking the expectations of equation 2.50 gives

$$E[x_m] = \lambda_\psi r \cos \psi \tag{2.52}$$
$$E[y_m] = \lambda_\psi r \sin \psi \tag{2.53}$$

where $\lambda_\psi = E[\cos w_\psi]$. The conversion in equation 2.50 is biased if $\lambda_\psi \neq 1$. An unbiased transformation can be given when $\lambda_\psi \neq 0$ as

$$x_m = \lambda_\psi^{-1} r_m \cos \psi_m \tag{2.54}$$
$$y_m = \lambda_\psi^{-1} r_m \sin \psi_m \tag{2.55}$$

The true covariance of transformed measurements in equations 2.52 and 2.53 are given as:

$$R_t^{11} = \lambda_\psi^{-1}(r^2 + \sigma_r^2)E[\cos^2 \psi_m] - r^2 \cos^2 \psi \tag{2.56}$$
$$R_t^{22} = \lambda_\psi^{-1}(r^2 + \sigma_r^2)E[\sin^2 \psi_m] - r^2 \sin^2 \psi \tag{2.57}$$
$$R_t^{12} = \lambda_\psi^{-1}(r^2 + \sigma_r^2)E[\sin^2 \psi_m \cos^2 \psi_m] - r^2 \sin \psi \cos \psi \tag{2.58}$$

This true covariance R_t is unavailable because the true range and bearing are unavailable in practice. Therefore in paper [13] an approximate covariance are presented where the

true values are not included in the computation of the transformed covariance matrix:

$$R_p^{11} = (\lambda_\psi^{-2} - 2) r_m^2 \cos^2 \psi_m + \frac{1}{2}(r_m^2 + \sigma_r^2)(1 + \lambda'_\psi \cos 2\psi_m) \tag{2.59}$$

$$R_p^{22} = (\lambda_\psi^{-2} - 2) r_m^2 \sin^2 \psi_m + \frac{1}{2}(r_m^2 + \sigma_r^2)(1 - \lambda'_\psi \cos 2\psi_m) \tag{2.60}$$

$$R_p^{12} = (\lambda_\psi^{-2} - 2) r_m^2 \cos \psi_m \sin \psi_m + \frac{1}{2}(r_m^2 + \sigma_r^2)\lambda'_\psi \sin 2\psi_m \tag{2.61}$$

where $\lambda_\psi = E[\cos w_\psi]$ and $\lambda'_\psi = E[cos 2 w_\psi]$. The superscript of R_p denotes indecies in the covariance matrix. For more details, see [13].

2.6.2 3D Measurements

As in for the 2D case, the transformation can be done for the 3D case. The measurements are:

$$r_m = r + w_r \tag{2.62}$$

$$\psi_m = \psi + w_\psi \tag{2.63}$$

$$\upsilon_m = \upsilon + w_\upsilon \tag{2.64}$$

where r, ψ and υ are true range, azimuth angle, and elevation angle of the target. The unbiased transformation are given as:

$$x_m = \lambda_\psi^{-1} \lambda_\upsilon^{-1} r_m \cos \psi_m \cos \upsilon \tag{2.65}$$

$$y_m = \lambda_\psi^{-1} \lambda_\upsilon^{-1} r_m \sin \psi_m \cos \upsilon \tag{2.66}$$

$$z_m = \lambda_\upsilon^{-1} R_m \sin \upsilon_m \tag{2.67}$$

The covariance matrix R_m then takes the form:

$$\begin{aligned} R_m^{11} = \frac{1}{4}\lambda_\psi^{-2}\lambda_\upsilon^{-2}(r_m^2 + 2\sigma_r^2)(1 + (\lambda'_\psi)^2 \cos 2\psi_m)(1 + (\lambda'_\upsilon)^2 \cos 2\upsilon_m) \\ -\frac{1}{4}(r_m^2 + \sigma_r^2)(1 + \lambda'_\psi \cos 2\psi_m)(1 + \lambda'_\upsilon \cos 2\upsilon_m) \end{aligned} \tag{2.68}$$

$$\begin{aligned} R_m^{22} = \frac{1}{4}\lambda_\psi^{-2}\lambda_\upsilon^{-2}(r_m^2 + 2\sigma_r^2)(1 - (\lambda'_\psi)^2 \cos 2\psi_m)(1 + (\lambda'_\upsilon)^2 \cos 2\upsilon_m) \\ -\frac{1}{4}(r_m^2 + \sigma_r^2)(1 - \lambda'_\psi \cos 2\psi_m)(1 + \lambda'_\upsilon \cos 2\upsilon_m) \end{aligned} \tag{2.69}$$

$$\begin{aligned} R_m^{33} = \frac{1}{2}\lambda_\upsilon^{-2}(r_m^2 + \sigma_r^2)(1 - (\lambda'_\upsilon)^2 \cos 2\upsilon_m) \\ -\frac{1}{2}(r_m^2 + \sigma_r^2)(1 - \lambda'_\upsilon \cos 2\upsilon_m) \end{aligned} \tag{2.70}$$

$$R_m^{12} = \frac{1}{4}\lambda_\psi^{-2}\lambda_\upsilon^{-2}(\lambda'_\psi)^2(r_m^2 + 2\sigma_r^2)\sin 2\psi_m(1 + (\lambda'_\psi)^2\cos 2\upsilon_m) \\ -\frac{1}{4}\lambda'_\psi(r_m^2 + \sigma_r^2)\sin 2\psi_m(1 + \lambda'_\upsilon\cos 2\upsilon_m) \tag{2.71}$$

$$R_m^{13} = \frac{1}{2}\lambda_\psi\lambda_\upsilon^{-2}(\lambda'_\psi)^2(r_m^2 + 2\sigma_r^2)\cos\psi_m\sin 2\upsilon_m \\ -\frac{1}{2}\lambda_\psi\lambda'_\upsilon(r_m^2 + \sigma_r^2)\cos\psi_m\sin 2\upsilon_m \tag{2.72}$$

$$R_m^{23} = \frac{1}{2}\lambda_\psi\lambda_\upsilon^{-2}(\lambda'_\upsilon)^2(r_m^2 + 2\sigma_r^2)\sin\psi_m\sin 2\upsilon_m \\ -\frac{1}{2}\lambda_\psi\lambda'_\upsilon(r_m^2 + \sigma_r^2)\sin\psi_m\sin 2\upsilon_m \tag{2.73}$$

where $\lambda_\psi = E[\cos w_\psi]$, $\lambda'_\psi = E[\cos 2w_\psi]$,$\lambda_\upsilon = E[\cos w_\upsilon]$ and $\lambda'_\upsilon = E[\cos 2w_\upsilon]$. These factors are determined from the distribution of the azimuth and elevation noise. If that is zero mean gaussian noise the factors are:

$$\lambda_\psi = E[\cos w_\psi] = e^{-\sigma_\psi^2/2} \tag{2.74}$$

$$\lambda_\psi = E[\cos 2w_\psi] = e^{-2\sigma_\psi^2/2} \tag{2.75}$$

$$\lambda_\upsilon = E[\cos w_\upsilon] = e^{-\sigma_\upsilon^2/2} \tag{2.76}$$

$$\lambda_\upsilon = E[\cos 2w_\upsilon] = e^{-2\sigma_\upsilon^2/2} \tag{2.77}$$

Chapter 3

Linear Filtering

Filtering are based on estimating the state vector at the current time where all passed measurements are known. The term "filter" comes from the fact that it "filters" out noise or an undesired signal. Prediction estimates the state vector at a future time [1]. The state vector $\underline{x}$ optimally contains all needed information of the system required to desribe it. This vector is implemented in a model of the system describing how the states develop over time. For example in tracking the state vector could be position and velocity of the target, giving $\underline{x} = \begin{bmatrix} p & v \end{bmatrix}^T$, where p is the position and v is the velocity. Many systems can contain hidden states and measured variables, estimation of the state vector can then obtain better state knowledge [22]. There will always be unmodeled dynamics, because of these unknown properties of a physical system. This can be handled by treating the unmodeled dynamics as noise. The noise can then be described by a stochastic process, such as additive white gaussian noise. A stochastic process includes random variables or stochastic variables that are defined by its probability density function (pdf).

The Kalman filter is a recursive linear state estimator. Arthur Gelb [1] describes a recursive filter as a filter which there is no need to store past measurements for the intention of computing present estimates. The recursive Kalman filter is a stochastic extension of least squares methods and Bayes estimation. It consists of five equations, two for time update, the predictor step, and three for measurement update, the corrector step. To implement the Kalman filter one need some knowledge of the system to be estimated, with its measurements, or observations. In the following section the system equations will be described with its properties. This then leeds into the presentation of the recursive linear Kalman filter. In appendix A the derivation of the Kalman filter equations are presented.

3.1 The Discrete-Time Linear Kalman Filter

Consider the linear stochastic discrete time target process equation

$$\underline{x}_{k+1} = \Phi_k \underline{x}_k + \Lambda_k \underline{u}_k + \Gamma_k \underline{v}_k \tag{3.1}$$

with measurements equation

$$\underline{z}_k = H_k \underline{x}_k + \underline{w}_k \tag{3.2}$$

where $\underline{x}_k \in \mathbb{R}^{n_x \times 1}$ is the system state vector at time t_k. $\Phi_k \in \mathbb{R}^{n_x \times n_x}$ is the transition matrix for the system equation, and allows calculation of the state vector at some time t, given complete knowledge of the state vector at t_0, in the absence of forcing functions [1]. $\underline{u}_k \in \mathbb{R}^{n_u \times 1}$ is a sequence of control inputs, where $\Lambda_k \in \mathbb{R}^{n_x \times n_u}$ binds the inputs to the system states. $\underline{v}_k \in \mathbb{R}^{n_v \times 1}$ is the process noise, assumed white with known covariance matrix Q and zero mean, where $\Gamma_k \in \mathbb{R}^{n_z \times n_z}$ binds the process noise to the process state vector.

In equation 3.2 $\underline{z}_k \in \mathbb{R}^{n_z}$ is the measurements vector at time t_k, where $H_k \in \mathbb{R}^{n_z \times n_x}$ is the observation matrix, which gives a linear relationship to the system state. $\underline{w}_k \in \mathbb{R}^{n_z}$ is the measurement noise, assumed to be white with known covariance matrix R, zero mean, and zero cross correlation with the process noise.

The noise processes $\underline{v}_k$ and $\underline{w}_k$ have the following properties [4]:

$$\begin{array}{ll} \underline{v}_k \sim N(0, Q_k) & E[v_k w_j^T] = Q_k \delta_{kj} \\ \underline{w}_k \sim N(0, R_k) & E[w_k w_j^T] = R_k \delta_{kj} \\ E[\underline{v}_k] = \underline{0} & E[w_k v_j^T] = \underline{0} \\ E[\underline{w}_k] = \underline{0} & E[\underline{x}_0 \underline{v}_k] = \underline{0} \\ & E[\underline{x}_0 \underline{w}_k] = \underline{0} \end{array} \tag{3.3}$$

where δ_{kj} is the Kronecker delta function

$$\delta_{kj} = \begin{cases} 1 & \text{if } k = j \\ 0 & \text{if } k \neq j \end{cases} \tag{3.4}$$

and $\underline{x}_0$ is the initial state value, meaning that the process noises have zero cross correlation with the initial values.

The target's state vector may be estimated using the Kalman filter equations (see Appendix A for computation of the Kalman filter equations):

$$\bar{\underline{x}}_{k+1} = \Phi_k \hat{\underline{x}}_k + \Lambda_k \underline{u}_k \tag{3.5}$$

$$\bar{P}_{k+1} = \Phi_k \hat{P}_k \Phi_k + \Gamma_k Q_k \Gamma_k^T \tag{3.6}$$

$$\hat{\underline{x}}_k = \bar{\underline{x}}_k + K_k(\underline{z}_k - H_k \bar{\underline{x}}_k) \tag{3.7}$$

$$\hat{K}_k = \bar{P}_k H_k^T (H_k \bar{P}_k H_k^T + R_k)^{-1} \tag{3.8}$$

$$\hat{P}_k = (I - K_k H_k) \bar{P}_k \tag{3.9}$$

This is a recursive linear state estimator, which uses a linear feedback of the innovation error to compute new estimates. The innovation error is the deviation $\underline{z}_k - H_k \bar{\underline{x}}_k$, accordently the error between the measurement and the estimated measurement. It also calculates the covariance matrix P, which describes the quality of the estimate, and how the states are statistically coupled. The Kalman filter are made up from a predictor step $\bar{\underline{x}}_{k+1}$, the time update, and a corrector step $\hat{\underline{x}}_k$, the measurement update. K_k is the Kalman gain.

Ristic, Arulampalama and Gordon in [2] mentions that the Kalman filter assumes that the posterior density for every time step t_k is Gaussian and therefore exactly characterized

by two parameters, its mean and covariance. Further they say that if $p(x_k|z_k)$ is Gaussian, it can be proved that $p(x_{k+1}|z_{k+1})$ is also Gaussian, provided that the assumesions 3.3 holds, and that process and measurement equations 3.1 and 3.2 are linear. The Kalman filter recursively computes the mean and covariance of the probability density function (pdf) $p(x_k|z_k)$, which is the optimal solution to the tracking problem. No other algorithm can do better than the Kalman filter in the linear Gaussian case [1, 4, 2].

3.1.1 Implementation of the Linear Kalman Filter

The algorithm for implementing the linear Kalman filter can be summarized as in Algorithm 1 [4].

Algorithm 1 Linear Kalman filter

Require: $\underline{x}_{k+1} = \Phi_k \underline{x}_k + \Lambda_k \underline{u}_k + \Gamma_k \underline{v}_k$ and $\underline{z}_k = H_k \underline{x}_k + \underline{w}_k$

Initialize with

$\hat{\underline{x}}_0 = E(\underline{x}_0)$ $\hat{P}_0 = E[(\underline{x}_0 - \hat{\underline{x}}_0)(\underline{x}_0 - \hat{\underline{x}}_0)^T]$

1: **for** k=1,2... **do**
2: **if** Time update **then**
3: $\bar{\underline{x}}_{k+1} = \Phi_k \hat{\underline{x}}_k + \Lambda_k \underline{u}_k$
4: $\bar{P}_{k+1} = \Phi_k \hat{P}_k \Phi_k + \Gamma_k Q_k \Gamma_k^T$
5: **end if**
6: **if** Measurement update **then**
7: $\hat{K}_k = \bar{P}_k H_k^T (H_k \bar{P}_k H_k^T + R_k)^{-1}$
8: $\hat{\underline{x}}_k = \bar{\underline{x}}_k + K_k(\underline{z}_k - H_k \bar{\underline{x}}_k)$
9: $\hat{P}_k = (I - K_k H_k)\bar{P}_k$
10: **end if**
11: **end for**

As we can see from Algorithm 1, the Kalman filter initialization requires a model of the system that are to be estimated, with its measurements. These are given as in equations 3.1 and 3.2 with the properties from 3.3. But as Dan Simon in [4] states, the implementation of the Kalman filter on a real system may not work as in the theory. That is mainly because the Kalman filter assumes that no model mismatch occurs. It assumes that Φ, Q, H and R are completely known, and that the noise $\underline{v}$ and $\underline{w}$ are white, zero mean and uncorrelated. If any of those assumptions do not hold in the real world, as they never do, the Kalman filter theory may not work. But the theory are robust, so it gives good estimates in most of the real problems. But the designer may do implementations to increase the performance. One may increase arithmetic precision, so that the filter more closely match the analog theory. One may use time on initialize the covariance matrices, to avoid large changes. The use of fictitous process noise, ecpecially for estimating constants are easy to implement and gives great improvements. The process noise is a way to tell the filter that you do not "trust" your model, and the filter uses more weight on the measurements than the process model.

Chapter 4

Non-Linear Filtering

In the previous section the Kalman filter equations for estimating the system state $\underline{x}$ for a linear system where presented. Unfortunately linear systems do not exist in the real world. But some systems may behave close to a linear system over small time intervals so that linear filtering can give satisfactory results. However one may come accross non-linear systems that do not behave linear in any way, and a non-linear filter approach are necessary.

> It appears that no particular approximate filter is consistently better than any other, though (...) any nonlinear filter is better than a strictly linear one.

Lawrence Schwartz and Edwin Stear [19].

4.1 Extended Kalman Filter

The extended Kalman filter extends the linear Kalman filter approach and gives good estimation results for non-linear systems. This filter is the most used non-linear filter for the past decades [4]. It is based directly on the linear Kalman filter, but in addition, the idea of the extended Kalman filter (EKF) is to linearize the non-linear system around the Kalman filter estimate. The time update and the measurement update are implemented in theire non-linear forms, but for the covariance matrix, the linearizations are needed.

4.1.1 The discrete-time extended Kalman filter

Consider the non-linear system model:

$$\underline{x}_{k+1} = f(\underline{x}_k, \underline{u}_k) + \underline{v}_k \tag{4.1}$$

$$\underline{z}_k = h(\underline{x}_k) + \underline{w}_k \tag{4.2}$$

To linearize the system around the state estimate we use their Jacobians and obtains

$$F_k = \left. \frac{\delta f}{\delta \underline{x}^T} \right| \underline{\hat{x}} \tag{4.3}$$

$$\Gamma_k = \left.\frac{\delta f}{\delta \underline{v}^T}\right|_{\underline{\hat{x}}} \tag{4.4}$$

and

$$H_k = \left.\frac{\delta h}{\delta \underline{x}^T}\right|_{\underline{\bar{x}}} \tag{4.5}$$

This gives us a linear state-space system and a linear measurement equation that we use to compute the predicted covariance $\bar{P}_{k+1}$. The predicted state estimate can be computed directly by use of $\underline{f}(\cdot)$. We get the extended Kalman filter equations

$$\underline{\bar{x}}_{k+1} = f(\underline{\hat{x}}_k, \underline{u}_k) \tag{4.6}$$
$$\bar{P}_{k+1} = F_k \hat{P}_k F_k^T + \Gamma_k Q_k \Gamma_k \tag{4.7}$$
$$\underline{\hat{x}}_k = \underline{\bar{x}}_k + K_k(\underline{z}_k - h(\underline{\bar{x}}_k)) \tag{4.8}$$
$$\hat{K}_k = \bar{P}_k H_k^T (H_k \bar{P}_k H_k^T + R_k)^{-1} \tag{4.9}$$
$$\hat{P}_k = (I - K_k H_k)\bar{P}_k \tag{4.10}$$

4.1.2 Implementation of the Extended Kalman Filter

The algorithm for implementing the extended Kalman filter can be summarized as in algorithm 2 [4]

Algorithm 2 Extended Kalman filter

Require: $\underline{x}_{k+1} = f(\underline{x}_k, \underline{u}_k, \underline{v}_k)$ and $\underline{z}_k = h(\underline{x}_k, \underline{w}_k)$

Initialize with

$\underline{\hat{x}}_0 = E(\underline{x}_0)$ $\hat{P}_0 = E[(\underline{x}_0 - \underline{\hat{x}}_0)(\underline{x}_0 - \underline{\hat{x}}_0)^T]$

1: **for** k=1,2... **do**
2: **if** Time update **then**
3: $F_k = \frac{\delta f}{\delta \underline{x}^T} \mid_{\underline{\hat{x}}}$
4: $\Gamma_k = \frac{\delta f}{\delta \underline{v}^T} \mid_{\underline{\hat{x}}}$
5: $\underline{\bar{x}}_{k+1} = f(\underline{\hat{x}}_k, \underline{u}_k)$
6: $\bar{P}_{k+1} = F_k \hat{P}_k F_k^T + \Gamma_k Q_k \Gamma_k$
7: **end if**
8: **if** Measurement update **then**
9: $H_k = \frac{\delta h}{\delta \underline{x}^T} \mid_{\underline{\bar{x}}}$
10: $\hat{K}_k = \bar{P}_k H_k^T (H_k \bar{P}_k H_k^T + R_k)^{-1}$
11: $\underline{\hat{x}}_k = \underline{\bar{x}}_k + K_k(z_k - h(\underline{\bar{x}}_k))$
12: $\hat{P}_k = (I - K_k H_k)\bar{P}_k$
13: **end if**
14: **end for**

4.1.3 EKF and its Flaws

Julier and Uhlmann had the limitations of the Extended Kalman filter in their minds when they presented the Sigma Point filter in [11]. They argumented, that the EKF simply use linearization on the non-linear transformations and substitutes Jacobians for the linear transformations in the KF equations, as we have seen. But the linearization approach can be poor if the process is highly non-linear. In the worst case the estimates can diverge. The linearization can only be done if the Jacobian matrix exists, which is not always the case since some systems for example can be discontinous, contain singularities or contain built-in discrete states.

Further Julier and Uhlmann based their arguments on that the calculations of Jacobians can be a error-prone process. This can leed to human coding errors and the need of difficult debugging particularly if you do not know which performance to expect. For example the deriatives of the transfomation from cartesian to spherical coordinates are quite troublesome to calculate. This further leeds to probabilities of typing errors when implementing it in a programming language.

In the next section the Unscented transformation are presented wich provides a mechanism for transforming mean and covariance information. And we shall see that the Unscented transformation gives better results transforming the mean and covariance of non-linear systems than the linearizations.

4.2 Sigma Point Filter

As mentined earlier, the extended Kalman filter is the most used non-linear filter. But it can be difficult to tune and may give unsatisfactory results since the system must be linearized around the estimated states, and may give poor estimates if the system is highly non-linear. The Sigma point filter was first presented by Simon J. Julier and Jeffrey K. Uhlmann in 1997 [20]. We shall now see why the Sigma point filter can give better estimates than the extended Kalman filter like it is presented in [4, 11]. Since the Sigma Point filter is fairly fresh in the history of Kalman filtering (especially for the author of this thesis), it will be given quite much interest and space in this thesis.

4.2.1 Means and covariances of non-linear transformations

The mean of a non-linear transformation.

Consider the non-linear transformation

$$y_1 = r\cos\Theta \tag{4.11}$$
$$y_2 = r\sin\Theta \tag{4.12}$$

We have a sensor that measures the range r and the angle Θ. We want to converte these measurements to rectangular coordinates y_1 and y_2. The coordinate transform can be written as

$$\underline{y} = \underline{h}(\underline{x}) \tag{4.13}$$

where $\underline{x}$ is defined as $\underline{x} = \begin{bmatrix} r & \Theta \end{bmatrix}^T$. Suppose x_1 (r) is a random variable with mean $\bar{r} = 1$ and standard deviation σ_r. Assume further that x_2 (Θ) is a random variable with mean $\bar{\Theta} = \frac{\pi}{2}$ and standard deviation σ_Θ, and that r and Θ are independent with symmetric probability density functions around the means like for example gaussian or uniform.

If we look at equations 4.11 and 4.12 would lead us to belive that y_1 has $mean = 0$ since $\bar{y}_1 = \bar{r}cos\bar{\Theta} = 0$ and y_2 has $mean = 1$ since $\bar{y}_2 = \bar{r}sin\bar{\Theta} = 1$. And if we linearize we get exactly this

$$\underline{\bar{y}} = E[\underline{h}(\underline{x})] \approx E[\underline{h}(\underline{x}) + \frac{\delta \underline{h}}{\delta \underline{x}} \mid_{\underline{\bar{x}}} (\underline{x} - \underline{\bar{x}})] = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \tag{4.14}$$

Let us now be more rigorous as in [4], and write that

$$r = \bar{r} + \tilde{r} \tag{4.15}$$
$$\Theta = \bar{\Theta} + \tilde{\Theta} \tag{4.16}$$

where $\tilde{r}$ and $\tilde{\Theta}$ are the deviation for r and Θ from mean. Then we have that

$$\bar{y}_1 = E[rcos\Theta] = E[(\bar{r} + \tilde{r})cos(\bar{\Theta} + \tilde{\Theta})] = \bar{r}cos\bar{\Theta} = 0 \tag{4.17}$$

Let us do the same for $\bar{y}_2$

$$\bar{y}_2 = E[rsin\Theta] = E[(\bar{r} + \tilde{r})sin(\bar{\Theta} + \tilde{\Theta})] = \bar{r}sin\bar{\Theta}E[cos\tilde{\Theta}] = E[cos\tilde{\Theta}] \tag{4.18}$$

see [4] for a more thorough computation. If we suppose that $\tilde{\Theta}$ is uniformly distributed between $\pm\Theta_m$, we have

$$\bar{y}_2 = E[cos\tilde{\Theta}] = \frac{sin\Theta_m}{\Theta_m} \tag{4.19}$$

We expected that $\bar{y}_2$ should be 1, but got a number less than 1 (since $\frac{sin\Theta_m}{\Theta_m} < 1 \quad \forall \quad \Theta_m > 0$ and $\lim_{\Theta_m \to 0} \frac{sin\Theta_m}{\Theta_m} = 1$). This reveals the problem with first order linearization. As an example from [11, 4] in figure 4.1 illustrates this.

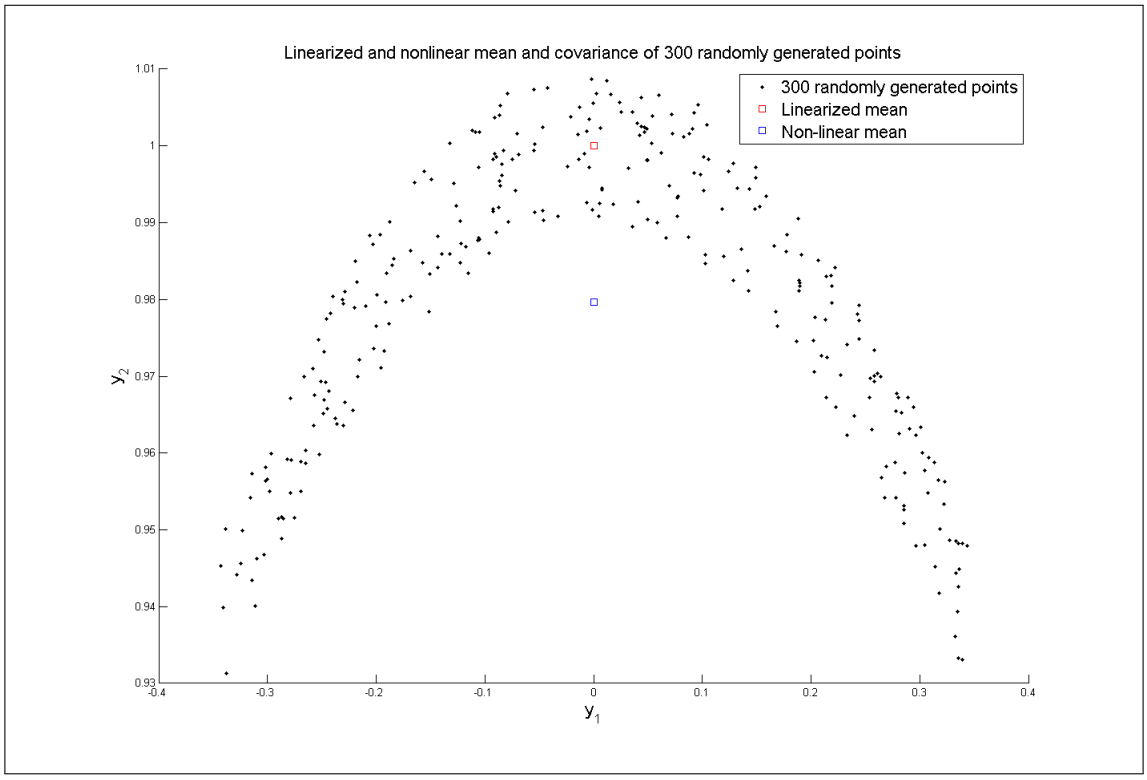

Figure 4.1: Linearized and nonlinear mean of 300 randomly generated points. $\tilde{r}$ uniformly distributed between ± 0.01 and $\tilde{\Theta}$ uniformly distributed between ± 0.35 radians.

The covariance of a nonlinear transfomation.

The covariance of $\underline{y}$ is defined by

$$P_{\underline{y}} = [(\underline{y} - \bar{\underline{y}})(\underline{y} - \bar{\underline{y}})^T] \tag{4.20}$$

Dan Simon in [4] gives a complete Taylor series expansion of the covariance $P_{\underline{y}}$ as

$$P_{\underline{y}} = HPH^T + E\left[\frac{D_{\tilde{\underline{x}}}\underline{h}(D^3_{\tilde{\underline{x}}}\underline{h})^T}{3!} + \frac{D^2_{\tilde{\underline{x}}}\underline{h}(D^2_{\tilde{\underline{x}}}\underline{h})^T}{2!2!} + \frac{D^3_{\tilde{\underline{x}}}h(D_{\tilde{\underline{x}}}\underline{h})^T}{3!}\right] + \left(\frac{D^2_{\tilde{\underline{x}}}\underline{h}}{2!}\right)\left(\frac{D^2_{\tilde{\underline{x}}}\underline{h}}{2!}\right)^2 \tag{4.21}$$

where

$$D^k_{\underline{\tilde{x}}}\underline{h} = \left(\sum_{i=1}^{n} \tilde{x}_i \frac{\partial}{\partial x_i}\right)^k \underline{h}(\underline{x})\Big|_{\underline{\bar{x}}} \tag{4.22}$$

In the Extended Kalman filter the only first part of this series expansion are being used to approximate the covariance of the estimation error. As an example, we have measurements given in the previous section as

$$\underline{y} = \underline{h}(\underline{x}) + \underline{w} \tag{4.23}$$

Then the covariance will be

$$P_y = HP_xH^T + R \tag{4.24}$$

where H is the partial derivative of $\underline{h}$ with respect of $\underline{x}$, and R is the covariance of $\underline{w}$. We have the same if we consider the process

$$\underline{x}_{k+1} = \underline{f}(\underline{x}_k) + \underline{v}_k \tag{4.25}$$

If we look at the equations for the Extended Kalman filter in equation 4.7 on page 20, we recognise the predicted covariance, $\bar{P}_{k+1}$ as

$$\bar{P}_{k+1} = F_k\hat{P}_kF^T + Q \tag{4.26}$$

where F is the partial derivative of $f(\underline{x})$ with respect of $\underline{x}$, and Q is the covariance of the process noise $\underline{v}_k$. These covariance approximations can give large errors if $\underline{h}(\underline{x})$ and $f(\underline{x})$ are highly non-linear.

Let us again look at the system in equations 4.11 and 4.12. A linear covariance approximation will indicate that $P_y \approx HP_{\underline{x}}H^T$ where

$$H = \frac{\partial h}{\partial \underline{x}}\,|_{\underline{x}=\underline{\bar{x}}} \tag{4.27}$$

$$= \begin{bmatrix} cos\Theta & -rsin\Theta \\ sin\Theta & rcos\Theta \end{bmatrix}_{\underline{x}=\underline{\bar{x}}} = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} \tag{4.28}$$

$$P_{\underline{x}} = E\left[\begin{bmatrix} r-\bar{r} \\ \Theta-\bar{\Theta} \end{bmatrix}\begin{bmatrix} r-\bar{r} \\ \Theta-\bar{\Theta} \end{bmatrix}^T\right] = \begin{bmatrix} \sigma_r^2 & 0 \\ 0 & \sigma_\Theta^2 \end{bmatrix} \tag{4.29}$$

which gives the covariance approximation

$$P_{\underline{y}} \approx HP_{\underline{x}}H^T = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}\begin{bmatrix} \sigma_r^2 & 0 \\ 0 & \sigma_\Theta^2 \end{bmatrix}\begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} = \begin{bmatrix} \sigma_\Theta^2 & 0 \\ 0 & \sigma_r^2 \end{bmatrix} \tag{4.30}$$

This can be written as [4]:

$$P_{\underline{y}} = E[(\underline{y} - \bar{\underline{y}})(\underline{y} - \bar{\underline{y}})^T] \tag{4.31}$$

$$= E\left[\begin{bmatrix} r\cos\Theta \\ r\sin\Theta - \frac{\sin\Theta_m}{\Theta_m} \end{bmatrix} \begin{bmatrix} r\cos\Theta \\ r\sin\Theta - \frac{\sin\Theta_m}{\Theta_m} \end{bmatrix}^T \right] \tag{4.32}$$

$$= \begin{bmatrix} r^2\cos^2\Theta & r^2\cos\Theta\sin\Theta - r\cos\Theta\frac{\sin\Theta_m}{\Theta_m} \\ r^2\cos\Theta\sin\Theta - r\cos\Theta\frac{\sin\Theta_m}{\Theta_m} & \left(r\sin\Theta - \frac{\sin\Theta_m}{\Theta_m}^2\right) \end{bmatrix} \tag{4.33}$$

We use the same assumptions as in the previous section saying that r and Θ are independent, r is uniformly distributed with $mean = 1$ and standard deviation σ_r, and $\sigma = \frac{\pi}{2} + \tilde{\Theta}$ with $\tilde{\Theta}$ uniformly distributed between $\pm\Theta_m$. Then we have that:

$$E(r^2) = 1 + \sigma_r^2 \tag{4.34}$$

$$E(cos^2\tilde{\Theta}) = \frac{1 - E(\cos 2\tilde{\Theta})}{2} \tag{4.35}$$

$$E(\cos 2\tilde{\Theta}) = \frac{\sin 2\Theta_m}{2\Theta_m} \tag{4.36}$$

$$E(\sin\Theta) = E(\cos\tilde{\Theta}) = \frac{\sin\Theta_m}{\Theta_m} \tag{4.37}$$

We now use this in equation 4.33 which gives

$$P_{\underline{y}} = \begin{bmatrix} \frac{1}{2}(1+\sigma_r^2)(1 - \frac{\sin 2\Theta_m}{2\Theta_m}) & 0 \\ 0 & \frac{1}{2}(1+\sigma_r^2)(1 + \frac{\sin 2\Theta_m}{2\Theta_m}) - \frac{\sin^2\Theta_m}{\Theta_m^2} \end{bmatrix} \tag{4.38}$$

This gives a 2 dimensional ellipse that defines the non-linear covariance which is illustrated in figure 4.2 [11, 4].

4.2.2 Unscented transformations

One problem with non-linear systems is that it can be difficult to transform a probability density function through a non-linear function. Extended Kalman filter uses a first order approximation of means and covariances, but in the previous section we saw that this do not always give satisfactory results. We shall now take a look at the unscented transformation and see that this will give better results.

An unscented transforamtion are based on two principles

1. It is easier to do a non-linear transformation of one point, than for an entire probability density function.

2. One can find a set of individual points in state space where sample probability density function approximates the true probability density function of a state vector.

The key of unscented transformation is to use these two prinsiples together. Suppose we know the mean $\bar{\underline{x}}$ and covariance P of a vector $\underline{x}$. We then find a set of deterministic vectors called sigma points where ensamble mean and covariance are equal to $\bar{\underline{x}}$ and P.

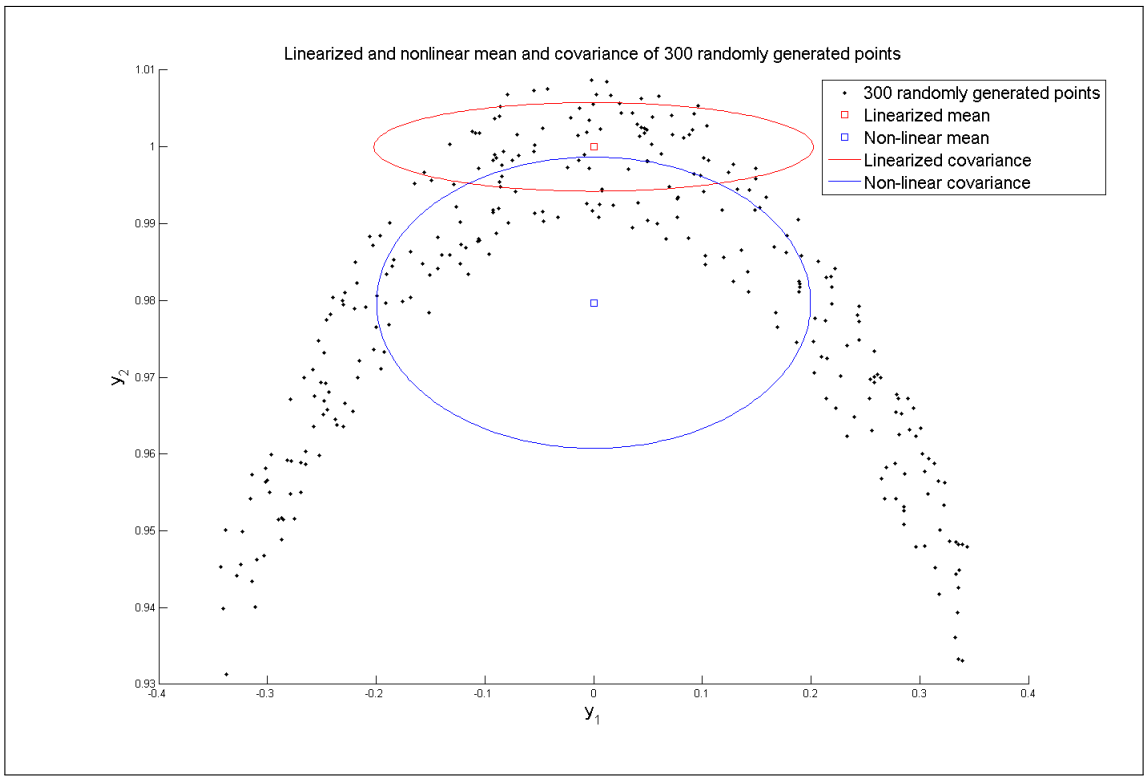

Figure 4.2: Linearized and nonlinear mean and covariance of 300 randomly generated points. $\tilde{r}$ uniformly distributed between ± 0.01 and $\tilde{\Theta}$ uniformly distributed between ± 0.35 radians.

Then we propagete these through the non-linear function $\underline{y} = \underline{h}(\underline{x})$ and gets transformed vectors. Ensamble mean and covariance of the transformed vectors will give a good estimate of the true mean and covariance of $\underline{y}$.

In [4], Dan Simon, proves that the approximated mean of $\underline{y}$ is the same as the true mean of $\underline{y}$ up to third order, while linearizing, as used in the extended Kalman filter, only matches up to first order. The unscented transformation can be done in the following manner:

1. We have a vector $\underline{x}$ with n_x elements and known mean $\bar{\underline{x}}$ and covariance P. We want to estimate the mean and covariance of $\underline{y} = \underline{h}(\underline{x})$, $\bar{\underline{y}}$ and $\bar{P}$.

2. Form 2n sigma point vectors $\underline{x}^{(i)}$

$$P = UU^T \quad U = [\underline{u}_1, \underline{u}_2, \ldots, \underline{u}_{n_x}] \tag{4.39}$$
$$\underline{x}^{(i)} = \bar{\underline{x}} + \sqrt{n_x}\underline{u}_i \text{ for } i = 1, 2, \ldots, n_x \tag{4.40}$$
$$\underline{x}^{(i+n_x)} = \bar{\underline{x}} - \sqrt{n_x}\underline{u}_i \text{ for } i = 1, 2, \ldots, n_x \tag{4.41}$$
$$\tag{4.42}$$

3. Propogate the sigma points through the non-linear function

$$\underline{y}^{(i)} = \underline{h}(\underline{x}^{(i)}) \text{ for } i = 1, 2, \ldots, 2n_x \tag{4.43}$$

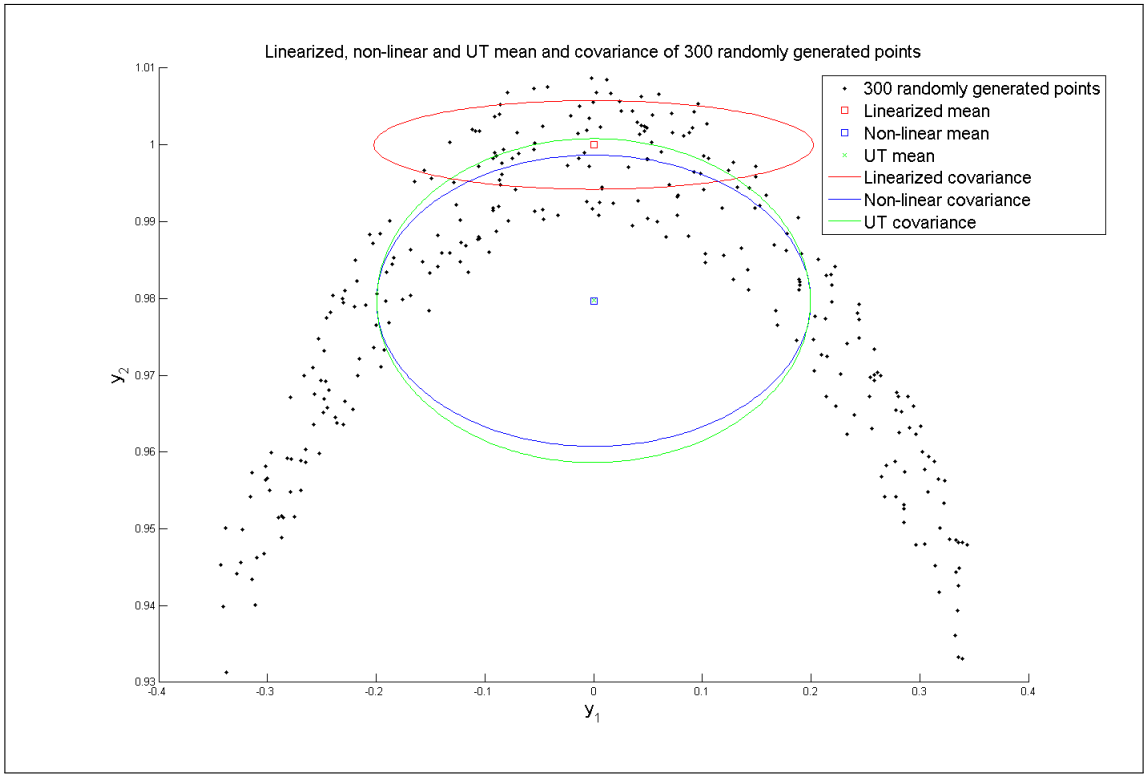

Figure 4.3: Linearized, nonlinear and UT mean and covariance of 300 randomly generated points. $\tilde{r}$ uniformly distributed between ± 0.01 and $\tilde{\Theta}$ uniformly distributed between ± 0.35 radians.

4. Approximate the mean and covariance of $\underline{y}$

$$\bar{\underline{y}} = \frac{1}{2n_x} \sum_{i=1}^{2n_x} \underline{y}^{(i)} \tag{4.44}$$

$$\bar{P} = \frac{1}{2n_x} \sum_{i=1}^{2n_x} (\underline{y}^{(i)} - \bar{\underline{y}})(\underline{y}^{(i)} - \bar{\underline{y}})^T \tag{4.45}$$

If this is used on the non-linear transformation in equations 4.11 and 4.11, the resulting mean and covariance are shown in figure 4.3 [11, 4]. The true mean and unscented mean are in fact plotted on top of each other. In comparison of the linearized mean, the unscented transformation made a more accurate transformation. The same conclusions can be made looking at the covariances. The unscented covariance is more accurate than the linearized.

4.2.3 The Sigma point filter

The unscented transformation in the previous section is the base line for the sigma point filter. The Sigma point filter can be generalized from the unscented transformation, and since the extended Kalman filter are based on linearization which gives poor results compared to the unscented transformation, we simply replace the extended Kalman filter equations with unscented transformation to get the Sigma point filter.

In [4], Dan Simon summarizes the Sigma point filter as follows

1. The discrete time, n_x states, non-linear system to be estimated are given by

$$\underline{x}_{k+1} = \underline{f}(\underline{x}_k, \underline{u}_k) + \underline{v}_k \tag{4.46}$$
$$\underline{z}_k = \underline{h}(\underline{x}_k) + \underline{w}_k \tag{4.47}$$
$$\underline{v}_k \sim N(0, Q_k) \tag{4.48}$$
$$\underline{w}_k \sim N(0, R_k) \tag{4.49}$$

2. Initialize the sigma point filter with

$$\hat{\underline{x}}_0 = E[\underline{x}_0] \tag{4.50}$$
$$\hat{P}_0 = E[(\underline{x}_0 - \hat{\underline{x}}_0)(\underline{x}_0 - \hat{\underline{x}}_0)^T] \tag{4.51}$$

3. The time update equations are given by

   (a) Calculate sigma points as in the previous section, but for each time update use the best guess for mean and covariance wich are $\hat{\underline{x}}_k$ and $\hat{P}_k$

$$\hat{P} = UU^T \quad U = [\underline{u}_1, \underline{u}_2, \ldots, \underline{u}_{n_x}] \tag{4.52}$$
$$\underline{x}_k^{(i)} = \hat{\underline{x}} + \sqrt{n_x}\underline{u}_i \text{ for } i = 1, 2, \ldots, n_x \tag{4.53}$$
$$\underline{x}_k^{(i+n_x)} = \hat{\underline{x}} - \sqrt{n_x}\underline{u}_i \text{ for } i = 1, 2, \ldots, n_x \tag{4.54}$$

   (b) Propogate the sigma points through the known non-linear function $f(\underline{x}_k, \underline{u}_k)$

$$\bar{\underline{x}}_{k+1}^{(i)} = f(\hat{\underline{x}}_k^{(i)}, \underline{u}_k) \tag{4.55}$$

   (c) Predict the state estimate and covariance

$$\bar{\underline{x}}_{k+1} = \frac{1}{2n_x}\sum_{i=1}^{2n_x} \bar{\underline{x}}_{k+1}^{(i)} \tag{4.56}$$
$$\bar{P}_{k+1} = \frac{1}{2n}\sum_{i=1}^{2n_x} (\bar{\underline{x}}_{k+1}^{(i)} - \bar{\underline{x}}_{k+1})(\bar{\underline{x}}_{k+1}^{(i)} - \bar{\underline{x}}_{k+1})^T + Q_k \tag{4.57}$$

4. The measurement update equations are given by

   (a) Propogate the sigma points through the known non-linear measurement function $\underline{h}(\underline{x}_k)$

$$\underline{z}_{k+1}^{(i)} = \underline{h}(\bar{\underline{x}}_{k+1}^{(i)}) \tag{4.58}$$

(b) Predict the state estimate, covariance and cross covariance

$$\bar{\underline{z}}_{k+1}^{(i)}\bar{P}_{zz}(k+1) = \frac{1}{2n_x}\sum_{i=1}^{2n_x}(\underline{z}_{k+1}^{(i)} - \bar{\underline{z}}_{k+1})(\underline{z}_{k+1}^{(i)} - \bar{\underline{z}}_{k+1})^T + R_{k+1} \tag{4.59}$$

$$\bar{P}_{xz}(k+1) = \frac{1}{2n_x}\sum_{i=1}^{2n_x}(\bar{\underline{x}}_{k+1}^{(i)} - \bar{\underline{x}}_{k+1})(\underline{z}_{k+1}^{(i)} - \bar{\underline{z}}_{k+1})^T \tag{4.60}$$

(c) Compute the estimate of the state vector and the covariance

$$K_{k+1} = \bar{P}_{xz}(k+1)\bar{P}_{zz}^{-1}(k+1) \tag{4.61}$$

$$\hat{P}_{k+1} = \bar{P}_{k+1} - K_{k+1}\bar{P}_{zz}(k+1)K_{k+1}^T \tag{4.62}$$

$$\hat{x}_{k+1} = \bar{\underline{x}} + K_{k+1}(\underline{z}_{k+1} - \bar{\underline{z}}_{k+1}) \tag{4.63}$$

Augmented state vector and covariance matrix.

The equations above applies only for additive white noise. If the process and measurement equations have noise that enters the system non-linearly like

$$\underline{x}_{k+1} = \underline{f}(\underline{x}_k, \underline{u}_k, \underline{v}_k) \tag{4.64}$$

$$\underline{z}_k = \underline{h}(\underline{x}_k, \underline{w}_k) \tag{4.65}$$

then we can handle the system as in [11] where Julier and Uhlmann presents an augmented state vector:

$$\hat{\underline{x}}_k^{'} = \begin{bmatrix} \hat{\underline{x}}_k \\ 0 \\ 0 \end{bmatrix} \tag{4.66}$$

and augmented covariance

$$\hat{P}_k' = \begin{bmatrix} \hat{P}_k & 0 & 0 \\ 0 & Q_k & 0 \\ 0 & 0 & R_{k+1} \end{bmatrix} \tag{4.67}$$

Then the sigma points will be:

$$\underline{x}_k^{'(i)} = \begin{bmatrix} \underline{x}_k^{(i)} \\ \underline{v}_k^{(i)} \\ \underline{w}_{k+1}^{(i)} \end{bmatrix} \tag{4.68}$$

Remove Q_k and R_{k+1} from equations 4.57 and 4.60. This will implement the noise that enters the system non-linearly.

Extended set of sigma points.

Sigma points chosen as in equations 4.39 to 4.42 will give a mean extremely close to the true transformed distribution, but the unscented transformation of the covariance estimate underestimates the true covariance. The reason for this is that the set of sigma points described are only accurate to the second order, and therefore gives the same order of accuracy as linearization [11]. Julier and Uhlmann presents an extended set of sigma

points, where one point are added which maintaines the mean of the set, but the sigma points must be scaled to remain the given covariance. This will give a different scaled sigma point set, which maintaines the the same mean and covariance, but different higher moments, and therfore gives a better representation of the covariance.

The extended sigma point set becomes [11]:

$$p = 2n_x + 1 \tag{4.69}$$

$$P = UU^T \quad U = [\underline{u}_1, \underline{u}_2, \ldots, \underline{u}_{n_x}] \tag{4.70}$$

$$\underline{x}^{(0)} = \bar{\underline{x}} \tag{4.71}$$

$$W^{(0)} = W^{(0)} \tag{4.72}$$

$$\underline{x}^{(i)} = \bar{\underline{x}} + \left(\sqrt{\frac{n_x}{1 - W^{(0)}}}\underline{u}_i\right) \text{ for } i = 1, 2, \ldots, n_x \tag{4.73}$$

$$\underline{x}^{(i+n_x)} = \bar{\underline{x}} - \left(\sqrt{\frac{n_x}{1 - W^{(0)}}}\underline{u}_i\right) \text{ for } i = 1, 2, \ldots, n_x \tag{4.74}$$

$$W^{(i)} = \frac{1 - W^{(0)}}{2n_x} \text{ for } i = 1, 2, \ldots, 2n_x \tag{4.75}$$

$$\underline{y}^{(i)} = h(\underline{x}^{(i)}) \text{ for } i = 1, 2, \ldots, 2n_x \tag{4.76}$$

$$\bar{\underline{y}} = \sum_{i=0}^{2n_x} W^{(i)}\underline{y}^{(i)} \tag{4.77}$$

$$\bar{P}_y = \sum_{i=0}^{2n_x} W^{(i)}(\underline{y}^{(i)} - \bar{\underline{y}})(\underline{y}^{(i)} - \bar{\underline{y}})^T \tag{4.78}$$

The weigth on the mean point $W^{(0)}$ are often chosen to be $\frac{1}{3}$ because it guarantees that some of the forth-order moments are the same as in the Gaussian case [11].

Scaled unscented transformation

Julier and Uhlmann presents a modified form of the non-linear transformation that can be used to reduce the effects of the higher order moments. Further they presents a modified sigma point set that affects the higher order moments the same way as an modified non-linear transformation [11].

The scaled sigma point set becomes:

$$p = 2n_x + 1 \tag{4.79}$$

$$P = UU^T \quad U = [\underline{u}_1, \underline{u}_2, \ldots, \underline{u}_{n_x}] \tag{4.80}$$

$$\kappa \geq 0; \quad 0 \leq \alpha \leq 1; \beta \geq 0; \quad \lambda = \alpha^2(n_x + \kappa) - n_x \tag{4.81}$$

$$\underline{x}^{(0)} = \bar{\underline{x}} \tag{4.82}$$

$$\underline{x}^{(i)} = \bar{\underline{x}} + \sqrt{n_x + \lambda}\underline{u}_i \text{ for } i = 1, 2, \ldots, n_x \tag{4.83}$$

$$\underline{x}^{(i+n_x)} = \bar{\underline{x}}\sqrt{n_x + \lambda}\underline{u}_i \text{ for } i = 1, 2, \ldots, n_x \tag{4.84}$$

$$W_m^{(0)} = \frac{\lambda}{n_x + \lambda} \tag{4.85}$$

$$W_c^{(0)} = \frac{\lambda}{n_x + \lambda} + (1 - \alpha^2 + \beta) \tag{4.86}$$

$$W_m^{(i)} = W_c^{(i)} = \frac{1}{2(n_x + \lambda)} \quad \text{for } i = 1, 2, \ldots, 2n_x \tag{4.87}$$

$$\underline{y}^{(i)} = h(\underline{x}^{(i)}) \text{ for } i = 1, 2, \ldots, 2n_x \tag{4.88}$$

$$\bar{\underline{y}} = \sum_{i=0}^{2n_x} W^{(i)}{}_m \underline{y}^{(i)} \tag{4.89}$$

$$\bar{P}_y = \sum_{i=0}^{2n_x} W_c^{(i)} (\underline{y}^{(i)} - \bar{\underline{y}})(\underline{y}^{(i)} - \bar{\underline{y}})^T \tag{4.90}$$

$$\tag{4.91}$$

Where $\lambda = \alpha^2(n_x + \kappa) - n_x$ is a scaling factor

- α, determines the spread of the sigma points around $\bar{\underline{x}}$.
- κ, is a secondary scaling parameter.
- β, is used to include prior knowledge of the distribution of $\underline{x}$. For the Gaussian case, $\beta = 2$ [11].

The algorithm for implementing the Sigma point filter with scaled unscented transformation can be summarized as in algorithm 3 [21].

Algorithm 3 Sigma Point Filter

Require: $\underline{x}_{k+1} = f(\underline{x}_k, \underline{u}_k) + \underline{v}_k$ and $\underline{z}_k = h(\underline{x}_k) + \underline{w}_k$

Initialize with

$\hat{\underline{x}}_0 = E(\underline{x}_0)$; $\hat{P}_0 = E[(\underline{x}_0 - \hat{\underline{x}}_0)(\underline{x}_0 - \hat{\underline{x}}_0)^T]$

for k=1,2... **do**

$\chi_k = \left[\hat{\underline{x}}_k, \hat{\underline{x}}_k + \left[chol((n_x + \lambda)\hat{P}_k), -chol((n_x + \lambda)\hat{P}_k)\right]\right]$

if Time update **then**

$\bar{\underline{x}}_{k+1}^{(i)} = f(\hat{\underline{x}}_k^{(i)}, \underline{u}_k)$

$\bar{\underline{x}}_{k+1} = \sum_{i=0}^{2n_x} W_m^{(i)} \bar{\underline{x}}_{k+1}^{(i)}$

$\bar{P}_{k+1} = \sum_{i=0}^{2n_x} W^{(i)} (\bar{\underline{x}}_{k+1}^{(i)} - \bar{\underline{x}}_{k+1})(\bar{\underline{x}}_{k+1}^{(i)} - \bar{\underline{x}}_{k+1})^T + Q_k$

$\bar{\underline{z}}_{k+1}^{(i)} = h(\bar{\underline{x}}_{k+1}^{(i)})$

$\bar{\underline{z}}_{k+1} = \sum_{i=1}^{2n_x} W^{(i)} \underline{z}_{k+1}^{(i)}$

end if

if Measurement update **then**

$\bar{P}_{zz}(k+1) = \sum_{i=0}^{2n_x} W^{(i)} (\underline{z}_{k+1}^{(i)} - \bar{\underline{z}}_{k+1})(\underline{z}_{k+1}^{(i)} - \bar{\underline{z}}_{k+1})^T + R_{k+1}$

$\bar{P}_{xz}(k+1) = \sum_{i=0}^{2n_x} W^{(i)} (\underline{x}_{k+1}^{(i)} - \bar{\underline{x}}_{k+1})(\underline{z}_{k+1}^{(i)} - \bar{\underline{z}}_{k+1})^T$

$K_{k+1} = \bar{P}_{xz}(k+1) \bar{P}_{z\bar{z}}^{-1}(k+1)$

$\hat{P}_{k+1} = \bar{P}_{k+1} - K_{k+1} \bar{P}_{zz}(k+1) K_{k+1}^T$

$\hat{\underline{x}}_{k+1} = \bar{\underline{x}}_{k+1} + K_{k+1}(\underline{z}_{k+1} - \bar{\underline{z}}_{k+1})$

end if

end for

{where $chol(A)$ is the cholesky factorization of the matrix A. χ is the matrix containing all the sigma point vectors $\underline{x}^{(i)}$ of the i'th column of χ. }

Chapter 5

Parameter Estimation

When the state vector includes parameters, or constants, that are needed to be estimated to be used in the dynamic process filter model, the state vector is augmented with the parameter to be estimated. Here parameter estimation using Kalman filters are presented in section 5.1. In addition frequency estimation using MUSIC is introduced in section 5.2 on the following page. Since the angular velocity is a parameter in the state vector in one of the cases described later in this thesis (Section 6.3 on page 43) the frequency is needed to be estimated. By experiments of estimating the process parameter, it was seen that the Kalman filters had problems estimating the angular velocity. This led to implementation of the frequency estimation algorithm MUSIC to assist the Kalman filters. Therefore the frequency estimation algorithm is introduced in this chapter of parameter estimation.

5.1 Kalman Filter Parameter Estimation

State estimation can in addition to estimate the states of a system, also be used to estimate the unknown parameters of a system [4]. If we are estimating both the states in $\underline{x}$ and a parameter vector $\underline{p}$ we have the process:

$$\underline{x}_{k+1} = \Phi_k(\underline{p})\underline{x}_k + \Lambda_k(\underline{p})\underline{u}_k + \Gamma_k(\underline{p})\underline{v}_k \tag{5.1}$$

$$\underline{z}_k = H_k\underline{x}_k + \underline{w}_k \tag{5.2}$$

$$\underline{p} = \text{ unknown parameter vector} \tag{5.3}$$

This can be extended so that the measurements $\underline{z}_k$ also depends on $\underline{p}$. To estimate $\underline{p}$, we just augments the state vector with the parameter, which yields:

$$\underline{x}'_k = \begin{bmatrix} \underline{x}_k \\ \underline{p}_k \end{bmatrix} \tag{5.4}$$

If the parameters in $\underline{p}$ are constants, it is necessary as discussed in the Section 3.1.1 on page 17 about Kalman filters, to model it as

$$\underline{p}_{k+1} = \underline{p}_k + \underline{v}_k \tag{5.5}$$

where $\underline{v}_k$ is a small artificial noise that allows modifications in the Kalman filter estimate. The augmented system model then becomes:

$$\underline{x}_{k+1} = \begin{bmatrix} \Phi_k(\underline{p}_k) + \Lambda_k(p_k)\underline{u}_k + \Gamma_k(p_k)\underline{v}_k \\ \underline{p}_k + \underline{v}_{p,k} \end{bmatrix} \tag{5.6}$$

$$= \underline{f}(\underline{x}'_k, \underline{u}_k, \underline{v}_k, \underline{v}_{p,k}) \tag{5.7}$$

$$\underline{z}_k = \begin{bmatrix} H_k & 0 \end{bmatrix} \begin{bmatrix} \underline{x}_k \\ \underline{p}_k \end{bmatrix} + \underline{v}_k \tag{5.8}$$

5.2 Multiple Signal Classification (MUSIC)

MUSIC is an eigenanalysis-based frequency estimation method of complex sinusoids observed in additive white noise [18]. That is, it is based on eigendecomposition of the estimated correlation matrix, or by forming a data matrix and then use singular value decomposition (SVD) to estimate eigenvalues and eigenvectors. MUSIC is called a subspace method, and are reported to give good estimations of frequencies. The theory presented here about MUSIC is to be found in [18]. Some more background could be read in [6].

A signal can be described as a linear combination of signals and noise, which gives a signal model as follows:

$$y_k = \sum_{i=1}^{p} A_i e^{j(2\pi f_i k + \phi_i)} + w_k \tag{5.9}$$

$$= \sum_{i=1}^{p} A_i s_{i,k} + w_k \tag{5.10}$$

where

y_k - measured signal at sample time k

A_i - amplitude of sinusoid i

ϕ_i - phase of sinusoid i

f_i - frequancy of sinusoid i

$w_k \sim N(0, \sigma_w^2)$ - additive white noise

p - number of sinusoids

$s_{i,k} = e^{j2\pi f_i k}$ - often called the stearing factor.

From this we can have a vector $\underline{y}$ that consists of the signal and the noise as:

$$\underline{y} = \underline{x} + \underline{w} \tag{5.11}$$

$$= S\underline{a} + \underline{w} \tag{5.12}$$

$$\Downarrow \tag{5.13}$$

$$\underline{x} = S\underline{a} \tag{5.14}$$

This gives the matrix corresponding to $\underline{x} = S\underline{a}$ as:

$$\begin{bmatrix} x_k \\ x_{k+1} \\ \vdots \\ x_{k+N-1} \end{bmatrix} = \begin{bmatrix} e^{j2\pi f_1 k} & e^{j2\pi f_2 k} & \cdots & e^{j2\pi f_p k} \\ e^{j2\pi f_1 (k+1)} & e^{j2\pi f_2 (k+1)} & \cdots & e^{j2\pi f_p (k+1)} \\ \vdots & \vdots & \ddots & \vdots \\ e^{j2\pi f_1 (k+N-1)} & e^{j2\pi f_2 (k+N-1)} & \cdots & e^{j2\pi f_p (k+N-1)} \end{bmatrix} \begin{bmatrix} A_1 e^{j2\pi\Phi_1} \\ A_2 e^{j2\pi\Phi_2} \\ \vdots \\ A_p e^{j2\pi\Phi_p} \end{bmatrix} \tag{5.15}$$

The correlation matrix R_{yy} can be described as the sum of signal correlation and noise correlation:

$$R_{yy} = R_{xx} + R_{ww} \tag{5.16}$$

where R_{xx} and R_{ww} are the signal and noise contributions respectively. This decomposition gives:

$$R_{yy} = SPS^H + \sigma_w^2 I \tag{5.17}$$

Where $\cdot^H$ is the hermitian transpose; the transpose of a complex conjugate. I is the identity matrix. P is the power of the sinusoids as:

$$P = \underline{a}\underline{a}^H = \begin{bmatrix} |A_1|^2 & 0 & \cdots & 0 & 0 \\ 0 & |A_2|^2 & \cdots & 0 & 0 \\ 0 & 0 & \ddots & 0 & 0 \\ \vdots & \vdots & \cdots & |A_i|^2 & \vdots \\ 0 & 0 & \cdots & 0 & 0 \end{bmatrix} \tag{5.18}$$

The correlation matrix can also be expressed on vector form as:

$$R_{yy} = \sum_{i=1}^{p} P_i \underline{s}_i \underline{s}_i^H + \sigma_w^2 I \tag{5.19}$$

where $\underline{s}$ is the coloumn vector of the matrix S. The eigenvalues and the eigenvectors of R_{xx} can be found by eigendecomposition:

$$R_{xx} = \sum_{i=1}^{N} \lambda_i \underline{v}_i \underline{v}_i^H = \sum_{i=1}^{p} \lambda_i \underline{v}_i \underline{v}_i^H \tag{5.20}$$

where λ_i is the eigenvalues and $\underline{v}_i$ is the eigenvectors. The noise correlation matrix has the same decomposition since the sum of all the cross products of the eigenvectors forms an identity matrix:

$$R_{ww} = \sigma_w^2 I = \sigma_w^2 \sum_{i=1}^{N} \underline{v}_k \underline{v}_k^H \tag{5.21}$$

R_{xx} and R_{ww} are infact not availeble, but R_{yy} can be estimated as we have seen:

$$R_{yy} = \sum_{i=1}^{p} \lambda_i \underline{v}_i \underline{v}_i^H + \sigma_w^2 \sum_{i=1}^{N} \underline{v}_i \underline{v}_i^H \tag{5.22}$$

$$= \sum_{i=1}^{p} (\lambda_i + \sigma_w^2) \underline{v}_i \underline{v}_i^H + \sigma_w^2 \sum_{i=p+1}^{N} \underline{v}_i \underline{v}_i^H \tag{5.23}$$

where $[\underline{v}_1, \ldots, \underline{v}_p]$ is the eigenvalues of the signal with corresponding eigenvalues $[\lambda_1, \ldots, \lambda_p]$. $[\underline{v}_{p+1}, \ldots, \underline{v}_N]$ is the eigenvalues of the noise with corresponding eigenvalues $[\lambda_{p+1}, \ldots, \lambda_N] = \sigma_w^2$.

In MUSIC, $[p+1, p+2, \ldots, N]$ noise eigenvectors can be treated, wich makes it possible to handle large amounts of data. The pseudo spectrum for MUSIC is defined as:

$$\hat{P}_{music}(f) = \frac{1}{\underline{s}^H(\sum_{i=p+1}^{N} \underline{v}_k \underline{v}_k^H)\underline{s}} \tag{5.24}$$

$$= \frac{1}{\underline{s}^H V V^H \underline{s}} \tag{5.25}$$

where $V = [\underline{v}_{p+1}, \ldots, \underline{v}_N]$ is the matrix of noise eigenvectors. $\hat{P}_{music}$ estimate is sharply peaked at the frequencies of the signal, which gives the MUSIC estimate.

Chapter 6

Mathematical Modelling

The modelling of the airplane trajectories and the falling body with the filter models are given in this chapter. The airplane motions are modelled for two different cases. The case with sine perturbations in the horizontal plane is presented in Section 6.1. Section 6.2 on page 42 presents the modelling of the airplane where it is in addition modelled a cosine perturbation in the vertical plane. Filter models for both filters (EKF and SPF) are introduced for the helix motion. Modelling of the falling body is presented in section 6.5.1 on page 55 with its filter models.

The main structure of this chapter will thus be simulation of the true trajectories for the airplane and the falling body. These trajectories are converted to spherical coordinates and added additive white noise to give the measurements for the filters that estimates the trajectories from these noisy measurements.

6.1 Horizontal Wave Motion

The horizontal wave motion scenario is modelled by the author as an airplane flying in a wave motion described from the **a**irplane frame ($\mathcal{F}_a$) as a sine perturbation horizontaly above the ground. The airplane is seen from a **r**adar frame ($\mathcal{F}_r$) where the radar is positioned in $\underline{p}_o^r = \begin{bmatrix} x & y & z \end{bmatrix}^T = \begin{bmatrix} 0 & 0 & 0 \end{bmatrix}^T$. $\mathcal{F}_a$ is positioned where the airplane enters the radar seight with x-direction pointing with the course of the airplane. The frames are illustrated in figure 6.1.

The wave motion seen from $\mathcal{F}_a$ is described by

$$y^a(t) = A\sin(\omega t) \tag{6.1}$$

where y^a is position seen from $\mathcal{F}_a$ at time t. A is the amplitude of the sine wave of 50 meters. ω is angular velocity given by $\omega = 2\pi f$ where f is a frequency of 0.1 Hz.

The airplane has a constant velocity in x-direction of 170 m/s, which gives

$$x^a(t) = v_x^a t \tag{6.2}$$

where x^a is the position seen from $\mathcal{F}_a$. v_x^a is the velocity in x direction seen from $\mathcal{F}_a$. The

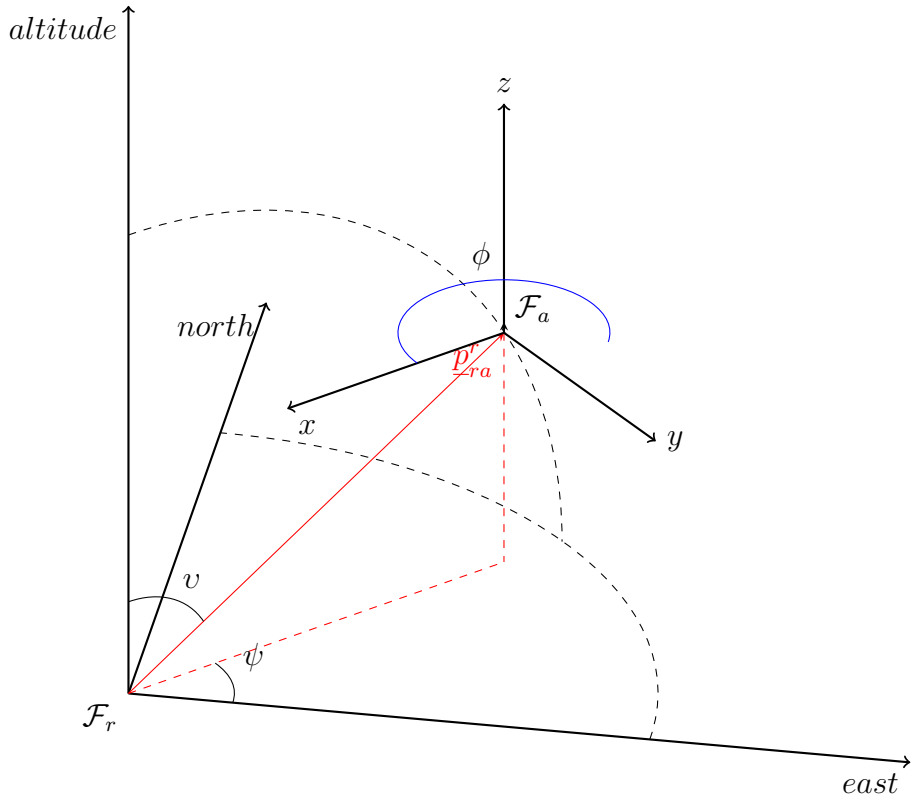

Figure 6.1: Radar frame, $\mathcal{F}_r$, and airplane frame, $\mathcal{F}_a$. $\underline{p}^r_{ra}$ is the vector from origo in $\mathcal{F}_r$ to $\mathcal{F}_a$. ψ is the azimuth angle and υ the elevation angle. Φ is the heading angle of the airplane, rotated around the z axis of $\mathcal{F}_a$, giving the relationship: $\underline{p}^r = \underline{p}^r_{ra} + R^r_a(\Phi)\underline{p}^a$.

airplane has a constant altidude, which yields

$$z^a(t) = 0 \tag{6.3}$$

where z^a is the position seen from $\mathcal{F}_a$.

By differentiation of the equations 6.1, 6.2 and 6.3 the velocities are given by:

$$v_x^a = \dot{x}^a = v_x^a \tag{6.4}$$
$$v_y^a = \dot{y}^a = A\omega\cos(\omega t) \tag{6.5}$$
$$v_z^a = \dot{z}^a = 0 \tag{6.6}$$

Twice differentiation gives the acceleration:

$$a_x^a = 0 \tag{6.7}$$
$$a_y^a = -A\omega^2\sin(\omega t) \tag{6.8}$$
$$a_z^a = 0 \tag{6.9}$$

To simulate the process in Matlab, a discrete state space model is needed on the form $\underline{x}_{k+1} = \underline{f}(\underline{x}_k)$. The state variables are then velocity and acceleration, and the state space model will describe the process completely by differential equations given a set of initial values. We can see that $\ddot{y}^a = -A\omega^2\sin(\omega t) = -\omega^2 y^a$. The state space model in continous time becomes

$$\dot{x}^a = \dot{x}_1^a = x_4^a \tag{6.10}$$
$$\dot{y}^a = \dot{x}_2^a = x_5^a \tag{6.11}$$
$$\dot{z}^a = \dot{x}_3^a = x_6^a \tag{6.12}$$
$$\dot{v}_x^a = \dot{x}_4^a = 0 \tag{6.13}$$
$$\dot{v}_y^a = \dot{x}_5^a = -\omega^2 x_2^a \tag{6.14}$$
$$\dot{v}_z^a = \dot{x}_6^a = 0 \tag{6.15}$$

Use of The improved Euler method with discretization time T, we have:

$$x_{1,k+1}^a = x_{1,k}^a + Tx_{4,k}^a \tag{6.16}$$
$$x_{2,k+1}^a = x_{2,k}^a + \frac{T}{2}(2x_{5,k}^a + T(-\omega^2 x_{2,k}^a)) \tag{6.17}$$
$$x_{3,k+1}^a = x_{3,k}^a + Tx_{6,k}^a \tag{6.18}$$
$$x_{4,k+1}^a = x_{4,k}^a \tag{6.19}$$
$$x_{5,k+1}^a = x_{5,k}^a + \frac{T}{2}(-2\omega^2 x_{2,k}^a + Tx_{5,k}^a) \tag{6.20}$$
$$x_{6,k+1}^a = x_{6,k}^a \tag{6.21}$$

The horizontal wave motion is illustrated in figure 6.2.

With use of a rotation matrix around the altitude axis (z-axis) the airplane motion

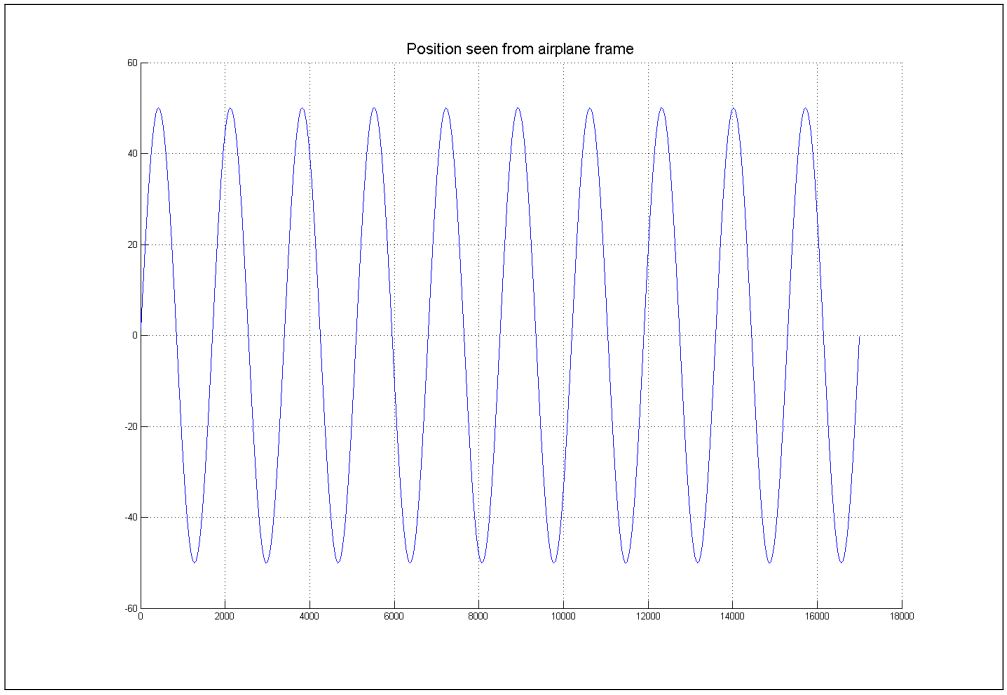

Figure 6.2: Horizontal wave motion of an airplane seen from $\mathcal{F}_a$ with amplitude of 50 meters and angular velocity of $\omega = 2\pi \times 0.1$.

can be seen from $\mathcal{F}_r$. The rotation matrix is given by [9]:

$$R_z(\phi) = \begin{bmatrix} \cos\phi & -\sin\phi & 0 \\ \sin\phi & \cos\phi & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{6.22}$$

where ϕ is the angle which to rotate around the z-axis. The frames can be studied in figure 6.1.

We now define a position vector as

$$\underline{p} = \begin{bmatrix} x \\ y \\ z \end{bmatrix} \tag{6.23}$$

The velocity vector will then become

$$\underline{v} = \dot{\underline{p}} \tag{6.24}$$

The system seen from $\mathcal{F}_r$ is then defined as in equation 2.42 on page 9 and we have that:

$$\dot{\underline{p}}^r = \underline{p}^r_{ra} + R^r_a(\phi)\dot{\underline{p}}^a \tag{6.25}$$

$$\dot{\underline{v}}^r = R^r_a(\phi)\dot{\underline{v}}^a \tag{6.26}$$

The horizontal wave motion seen from $\mathcal{F}_r$ is illustrated in figure 6.3.

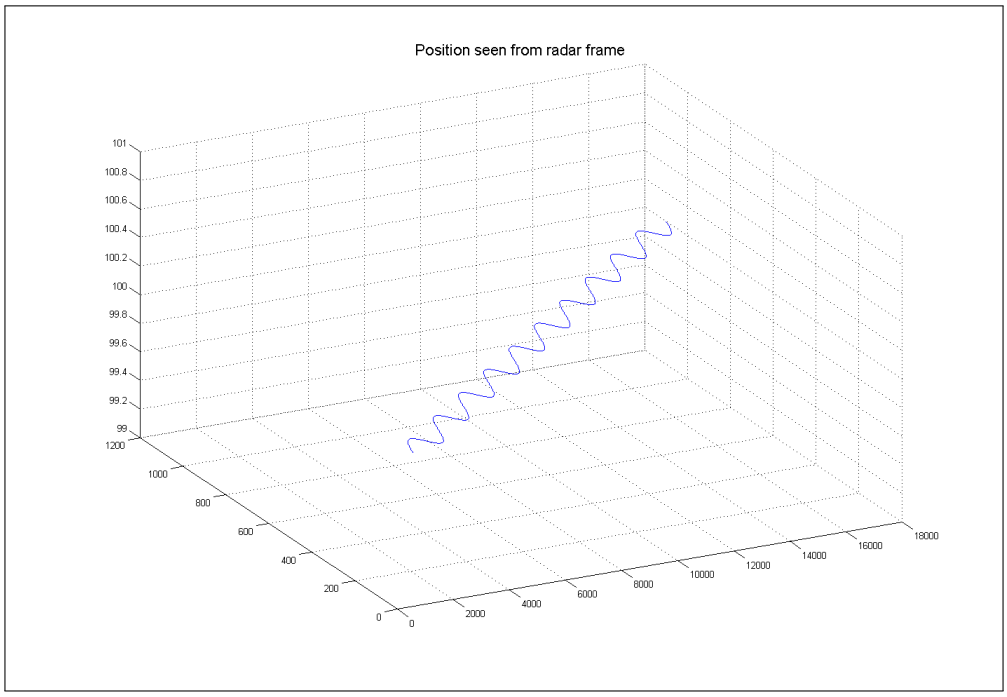

Figure 6.3: Horizontal wave motion of an airplane seen from $\mathcal{F}_r$ with amplitude of 50 meters and angular velocity of $\omega = 2\pi \times 0.1$. Heading straight towards the radar.

6.2 Helix Motion

The helix motion is described by both a sine perturbation in the horizontal plane and a cosine perturbation in the vertical plane. The position equations then becomes

$$x^a(t) = v_x^a t \tag{6.27}$$
$$y^a(t) = A\sin(\omega t) \tag{6.28}$$
$$z^a(t) = A\cos(\omega t) \tag{6.29}$$

By differentiation of the equations 6.29, the velocities are given by:

$$v_x^a = v_x^a \tag{6.30}$$
$$v_y^a = A\omega\cos(\omega t) \tag{6.31}$$
$$v_z^a = -A\omega\sin(\omega t) \tag{6.32}$$

Twice differentiation gives the acceleration:

$$a_x^a = 0 \tag{6.33}$$
$$a_y^a = -A\omega^2\sin(\omega t) \tag{6.34}$$
$$a_z^a = -A\omega^2\cos(\omega t) \tag{6.35}$$

To simulate the process in Matlab, a discrete state space model is needed on the form $\underline{x}_{k+1} = \underline{f}(\underline{x}_k)$. The state variables are then velocity and acceleration, and the state space model will describe the process completely by differential equations given a set of initial values. We can see that $\ddot{y}^a = -A\omega^2\sin(\omega t) = -\omega^2 y^a$, and that $\ddot{z}^a = -A\omega^2\cos(\omega t) = -\omega^2 z^a$. The state space model in continous time becomes

$$\dot{x}^a = \dot{x}_1^a = x_4^a \tag{6.36}$$
$$\dot{y}^a = \dot{x}_2^a = x_5^a \tag{6.37}$$
$$\dot{z}^a = \dot{x}_3^a = x_6^a \tag{6.38}$$
$$\dot{v}_x^a = \dot{x}_4^a = 0 \tag{6.39}$$
$$\dot{v}_y^a = \dot{x}_5^a = -\omega^2 x_2^a \tag{6.40}$$
$$\dot{v}_z^a = \dot{x}_6^a = -\omega^2 x_3^a \tag{6.41}$$

Use of The improved Euler method with discretization time T, we have:

$$\begin{array}{ll} x_{1,k+1}^a = x_{1,k}^a + Tx_{4,k}^a & x_{4,k+1}^a = x_{4,k}^a \\ x_{2,k+1}^a = x_{2,k}^a + \frac{T}{2}(2x_{5,k}^a + T(-\omega^2 x_{2,k}^a)) & x_{5,k+1}^a = x_{5,k}^a + \frac{T}{2}(-2\omega^2 x_{2,k}^a + Tx_{5,k}^a) \\ x_{3,k+1}^a = x_{3,k}^a + \frac{T}{2}(2x_{6,k}^a + T(-\omega^2 x_{3,k}^a)) & x_{6,k+1}^a = x_{6,k}^a + \frac{T}{2}(-2\omega^2 x_{3,k}^a + Tx_{6,k}^a) \end{array} \tag{6.42}$$

The helix motion is illustrated in figure 6.4. As in the horizontal wave motion case, we

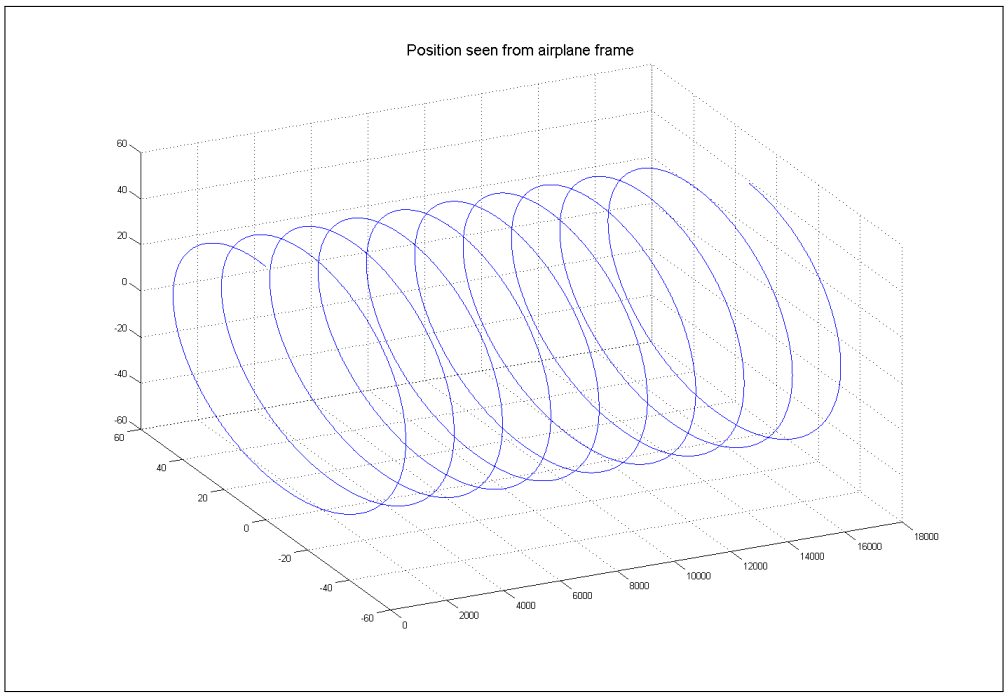

Figure 6.4: Helix motion of an airplane seen from $\mathcal{F}_a$ with amplitude of 50 meters and angular velocity of $\omega = 2\pi \times 0.1$.

can define

$$\dot{\underline{p}}^r = \underline{p}^r_{ra} + R^r_a(\phi)\dot{\underline{p}}^a \tag{6.43}$$

$$\dot{\underline{v}}^r = R^r_a(\phi)\dot{\underline{v}}^a \tag{6.44}$$

and get the process seen from $\mathcal{F}_r$ as in figure 6.5.

6.3 The Filter Models

In order to use the EKF with a non-linear process, the process have to be linearized around the state estimate using equation 4.3. This is used to compute the predicted covariance. As with the process, the measurements must be linearized using equation 4.5. The Sigma Point filter propogates the sigma point vectors through the non-linear functions for both process and measurements.

Because of the similarity of the filter models for the horizontal and helix case, only the filter models for the helix motion are described here. There are presented filter models for two cases, where the perturbations in horizontal and vertical plane are unknown and known. For the case where the perturbations are unknown, the process filter models are linear.

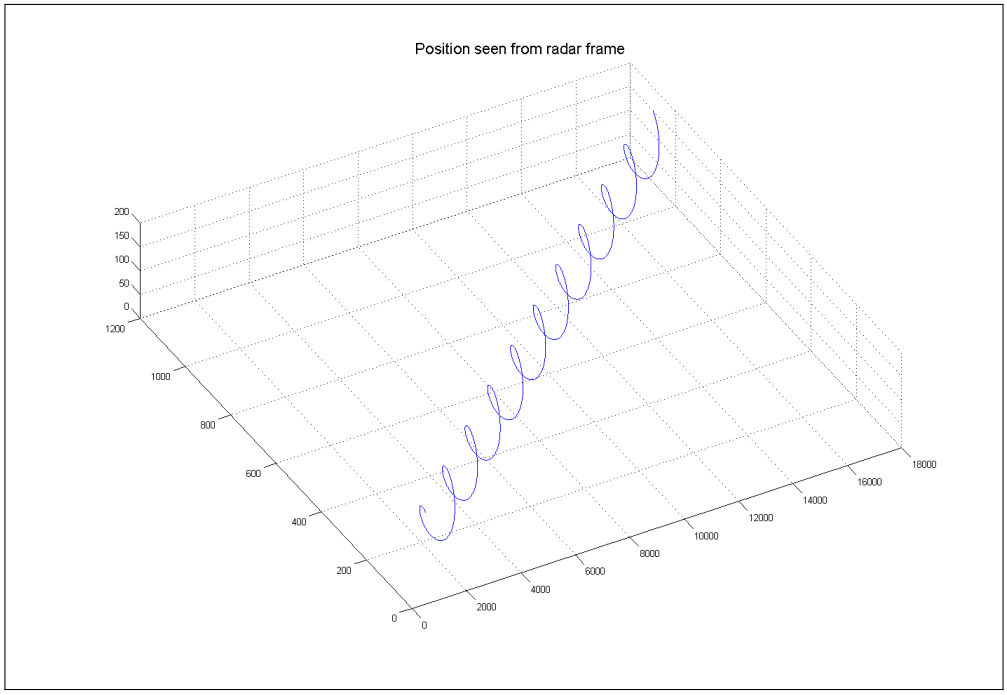

Figure 6.5: Helix motion of an airplane seen from $\mathcal{F}_r$ with amplitude of 50 meters and angular velocity of $\omega = 2\pi \times 0.1$. Heading straight towards the radar.

6.3.1 EKF with linear process model

In the case where the perturbations are unknown, only the measurements must be linearized. The measurements are non-linear on the form:

$$\underline{z}_k = \underline{h}(\underline{x}_k) + \underline{w}_k \tag{6.45}$$

where

$$\underline{z}_k = \begin{bmatrix} r_k \\ \psi_k \\ \upsilon_k \end{bmatrix} \tag{6.46}$$

Here, r is range, ψ is azimuth, the horizontal angular distance, and υ is elevation, the angular altitude from the radar, in a spherical coordinate system. They are related to the cartesian coordinate centered at the radar location. The conversion to spherical coordinates can be derived as in [5], and we have:

$$r = \sqrt{x^2 + y^2 + z^2} \tag{6.47}$$

$$\psi = \arctan \frac{y}{x} \tag{6.48}$$

$$\upsilon = \arctan \frac{\sqrt{x^2 + y^2}}{z} = \arccos \frac{z}{r} \tag{6.49}$$

The conversion from spherical to cartesian coordinates are:

$$x = r\sin\upsilon\cos\psi \tag{6.50}$$

$$y = r\sin\upsilon\sin\psi \tag{6.51}$$

$$z = r\cos\upsilon \tag{6.52}$$

By linearization we obtain the measurement matrix H

$$H = \begin{bmatrix} \frac{x}{\sqrt{x^2+y^2+z^2}} & \frac{y}{\sqrt{x^2+y^2+z^2}} & \frac{z}{\sqrt{x^2+y^2+z^2}} & 0 & 0 & 0 \\ \frac{-y}{x^2+y^2} & \frac{x}{x^2+y^2} & 0 & 0 & 0 & 0 \\ \frac{xz}{(x^2+y^2+z^2)\sqrt{x^2+y^2}} & \frac{yz}{(x^2+y^2+z^2)\sqrt{x^2+y^2}} & -\frac{x^2+y^2}{x^2+y^2+z^2} & 0 & 0 & 0 \end{bmatrix} \tag{6.53}$$

The process filter model in this case is linear in cartesian coordinates on the form $\dot{\underline{x}}(t) = F\underline{x}(t)$. The state vector is defined as $\underline{x} = \begin{bmatrix} \underline{p}^r & \underline{v}^r \end{bmatrix}^T$. In continous time we have:

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \\ \dot{x}_4 \\ \dot{x}_5 \\ \dot{x}_6 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \end{bmatrix} \tag{6.54}$$

Use of discritization with discritization time T [1], we have

$$\begin{bmatrix} x_{1,k+1} \\ x_{2,k+1} \\ x_{3,k+1} \\ x_{4,k+1} \\ x_{5,k+1} \\ x_{6,k+1} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & T & 0 & 0 \\ 0 & 1 & 0 & 0 & T & 0 \\ 0 & 0 & 1 & 0 & 0 & T \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_{1,k} \\ x_{2,k} \\ x_{3,k} \\ x_{4,k} \\ x_{5,k} \\ x_{6,k} \end{bmatrix} \tag{6.55}$$

These models are implemented in algorithm 2 for the extended Kalman filter. This will of course result in a linear time update of the filter. Estimating a non-linear trajectory, this may not be satisfactory especially if the system have infrequently measurement updates, and cause drifts between the measurement updates. But if one do not have any ideas of the dynamics of a true trajectory, a linear process filter modell may be the best choice.

6.3.2 SPF with linear process model

As for the EKF model the measurement model is non-linear on the form:

$$\underline{z}_k = \underline{h}(\underline{x}_k) + \underline{w}_k \tag{6.56}$$

where

$$\underline{z}_k = \begin{bmatrix} r_k \\ \psi_k \\ \upsilon_k \end{bmatrix} \tag{6.57}$$

The conversion to spherical coordinates are as in equations 6.51-6.52.

$$r = \sqrt{x^2 + y^2 + z^2} \tag{6.58}$$

$$\psi = \arctan \frac{y}{x} \tag{6.59}$$

$$\upsilon = \arctan \frac{\sqrt{x^2 + y^2}}{z} = \arccos \frac{z}{r} \tag{6.60}$$

The non-linear measurement equation becomes:

$$\underline{h} = \begin{bmatrix} r \\ \psi \\ \upsilon \end{bmatrix} \tag{6.61}$$

$$= \begin{bmatrix} \sqrt{x^{r2} + y^{r2} + z^{r2}} \\ \arctan \frac{y^r}{x^r} \\ \arccos \frac{z^r}{r} \end{bmatrix} \tag{6.62}$$

The linear process filter model are the same as in the EKF case in equation 6.55. These models are implemented in algorithm 3 for the sigma point filter.

6.3.3 EKF with non-linear process model

Here we will assume a helix motion using equations 6.36 to 6.41 on page 42, and add a new state for the angular velocity ω, a new state for heading angle ϕ and three new states for the origo of $\mathcal{F}_a$ seen from $\mathcal{F}_r$, that is $\underline{p}^r_{ra} = [p^r_{x,ra}, p^r_{y,ra}, p^r_{z,ra}]$. This gives a state vector on the form:

$$\underline{x} = \begin{bmatrix} \underline{p}^a \\ \underline{v}^a \\ \underline{p}^r_{ra} \\ \phi \\ \omega \end{bmatrix}^{11\times 1} \tag{6.63}$$

Now the state vector consists of 11 states, namely three for position $[p_x, p_y, p_z]^T$, three for velocity $[v_x, v_y, v_z]^T$ and the states mentioned above. All the states are being estimated in $\mathcal{F}_a$, and rotatet back to $\mathcal{F}_r$. The process filter model in discrete time becomes:

$$\begin{aligned} x^a_{1,k+1} &= x^a_{1,k} + T x^a_{4,k} \\ x^a_{2,k+1} &= x^a_{2,k} + T x^a_{5,k} \\ x^a_{3,k+1} &= x^a_{3,k} + T x^a_{6,k} \\ x^a_{4,k+1} &= x^a_{4,k} \\ x^a_{5,k+1} &= x^a_{5,k} - T x^{a2}_{11,k} \times x^a_{2,k} \\ x^a_{6,k+1} &= x^a_{6,k} - T x^{a2}_{11,k} \times x^a_{3,k} \end{aligned} \qquad \begin{aligned} x^r_{7,k+1} &= x^r_{7,k} \\ x^r_{8,k+1} &= x^r_{8,k} \\ x^r_{9,k+1} &= x^r_{9,k} \\ x^a_{10,k+1} &= x^a_{10,k} \\ x^a_{11,k+1} &= x^a_{11,k} \end{aligned} \tag{6.64}$$

The linearization of the filter model using equation 4.3 becomes

$$F = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -x_{11}^{a2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -2x_{11}^{a} \times x_{2}^{a} \\ 0 & 0 & x_{11}^{a2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -2x_{11}^{a} \times x_{3}^{a} \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{6.65}$$

F is discretized using $\Phi = e^{FT}$.

The measurement equation are modified from the previous section which gives a relationship between $\underline{p}^r$ and $\underline{p}^a$ as:

$$\underline{z} = \underline{h}(\underline{p}^r) + \underline{w} \tag{6.66}$$

$$= \underline{h}(\underline{p}_{ra}^r + R_a^r(\phi)\underline{p}^a) + \underline{w} \tag{6.67}$$

Here we can see that the measurement equation includes $\underline{p}_{ra}^r$, Φ and $\underline{p}^a$, this will improve the filter's ability to estimate just these mentioned variables, which is of great importance of the tracking problem in 3D where these are completely unknown. In addition, it is more desireble to have a filter model that does not only handle oscillations around the x axis. The sine wave and cosine wave that describes the helix can not be represented directly in $\mathcal{F}_r$. The reason for this is that the oscillations are described around the x axis. So if the airplane comes from, for example north east, 45 degrees above the x axis towards the radar, the equations of the helix are not applicable anymore. Thus, the state vector is represented and estimated in $\mathcal{F}_a$ and rotated back to $\mathcal{F}_r$. Because of this, the vector $\underline{p}_{ra}^r$ is needed to be estimated to define origo in $\mathcal{F}_a$ seen from $\mathcal{F}_r$. The heading angle Φ is needed to know the course of the airplane, and define the x axis of $\mathcal{F}_a$ that the airplane is oscillating around. Therefore the measurement equation 6.67 are used to give the relation between the measured position in $\mathcal{F}_r$ to the position in $\mathcal{F}_a$.

In the previous section $\underline{h}$ is the transformation from cartesian to spherical coordinates, but here we will for simplicity transform from spherical to cartesian coordinates using equations 2.65 to 2.67 and skip the non-linear transformation from cartesian to spherical coordinates in the measurement equation. This would have caused an even more troublesome differentiation of dense algebra as in the previous case. Thus, the measurements in cartesian coordinates are:

$$\underline{z} = \underline{p}^r + \underline{w} \tag{6.68}$$

$$= \underline{p}_{ra}^r + R_a^r(\Phi)\underline{p}^a + \underline{w} \tag{6.69}$$

By linearization of equation 6.69 we obtain the measurement matrix H

$$H = \begin{bmatrix} \cos\phi & -\sin\phi & 0 & 0 & 0 & 0 & 1 & 0 & 0 & -x^a \sin\phi - y^a \cos\phi & 0 \\ \sin\phi & \cos\phi & 0 & 0 & 0 & 0 & 0 & 1 & 0 & x^a \cos\phi - y^a \sin\phi & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix} \tag{6.70}$$

These models are implemented with algorithm 2 on page 20 to get the extended Kalman filter.

6.3.4 SPF with non-linear process model

In this case the sigma point vectors are propogated through the non-linear functions for the process and measurements directly, giving the process model as:

$$\begin{aligned} x^a_{1,k+1} &= x^a_{1,k} + T x^a_{4,k} \\ x^a_{2,k+1} &= x^a_{2,k} + T x^a_{5,k} \\ x^a_{3,k+1} &= x^a_{3,k} + T x^a_{6,k} \\ x^a_{4,k+1} &= x^a_{4,k} \\ x^a_{5,k+1} &= x^a_{5,k} - T x^{a2}_{11,k} \times x^a_{2,k} \\ x^a_{6,k+1} &= x^a_{6,k} - T x^{a2}_{11,k} \times x^a_{3,k} \end{aligned} \qquad \begin{aligned} x^r_{7,k+1} &= x^r_{7,k} \\ x^r_{8,k+1} &= x^r_{8,k} \\ x^r_{9,k+1} &= x^r_{9,k} \\ x^a_{10,k+1} &= x^a_{10,k} \\ x^a_{11,k+1} &= x^a_{11,k} \end{aligned} \tag{6.71}$$

and the measurement equation:

$$\underline{z} = \underline{h}(\underline{x}) \tag{6.72}$$

$$\underline{h} = \underline{p}^r_{ra} + R^r_a(\phi)\underline{p}^a \tag{6.73}$$

$$= \begin{bmatrix} \bar{x}^r_7 \\ \bar{x}^r_8 \\ \bar{x}^r_9 \end{bmatrix} + \begin{bmatrix} \cos x^a_{10} & -\sin x^a_{10} & 0 \\ \sin x^a_7 & \cos x^a_{10} & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x^a_1 \\ x^a_2 \\ x^a_3 \end{bmatrix} \tag{6.74}$$

For the Sigma Point filter it would have been a less error prone process using the transformation from cartesian to spherical coordinates in the measurement equation. The linearization is not needed, and equation 6.74 could just been inserted into the transfomation. But since the filters results shall be compared, a similar implementation are used.

6.3.5 Summary

Linear process filter model

The system model:

$$\left.\begin{array}{l} x^a(t) = v_x^a t \\ y^a(t) = A\sin(\omega t) \\ z^a(t) = A\cos(\omega t) \end{array}\right\} \underline{p}^a \qquad \begin{array}{l} \dot{\underline{p}}^r = \underline{p}_{ra}^r + R_a^r(\phi)\dot{\underline{p}}^a \\ \dot{\underline{v}}^r = R_a^r(\phi)\dot{\underline{v}}^a \end{array} \tag{6.75}$$

$$\begin{array}{l} \underline{z} = \underline{h}(\underline{p}^r) + \underline{w} \\ \underline{h} = \text{Transformation from cartesian to spherical coordinates} \end{array} \tag{6.76}$$

The filter models:

The state vector for both filters:

$$\underline{x} = \begin{bmatrix} \underline{p}^r \\ \underline{v}^r \end{bmatrix}^{6\times 1} \tag{6.77}$$

The process filter models:

$$\begin{array}{l} \dot{x}^r = v_x^r \\ \dot{y}^r = v_y^r \\ \dot{z}^r = v_z^r \\ \dot{v}_x^r = 0 \\ \dot{v}_y^r = 0 \\ \dot{v}_z^r = 0 \end{array} \tag{6.78}$$

The measurement model:

$$\begin{array}{l} \underline{z} \text{ is in spherical coordinates} \\ \underline{z} = \underline{h}(\underline{x}) \end{array} \qquad \underline{h} = \begin{bmatrix} \sqrt{x^{r,2} + y^{r,2} + z^{r,2}} \\ \arctan \frac{y^r}{x^r} \\ \arccos \frac{z^r}{\sqrt{x^{r,2}+y^{r,2}+z^{r,2}}} \end{bmatrix} \tag{6.79}$$

Non-linear process model

The system model:

$$\left.\begin{array}{l} y^a(t) = A\sin(\omega t) \\ x^a(t) = v_x^a t \\ y^a(t) = A\cos(\omega t) \end{array}\right\}\underline{p}^a \qquad \begin{array}{l} \underline{\dot{p}}^r = \underline{p}_{ra}^r + R_a^r(\phi)\underline{\dot{p}}^a \\ \underline{\dot{v}}^r = R_a^r(\phi)\underline{\dot{v}}^a \end{array} \tag{6.80}$$

$$\begin{array}{l} \underline{z} = \underline{h}(\underline{p}^r) + \underline{w} \\ \underline{h} = \text{Transformation from cartesian to spherical coordinates} \end{array} \tag{6.81}$$

The filter models:

The state vector for both filters:

$$\underline{x} = \begin{bmatrix} \underline{p}^a \\ \underline{v}^a \\ \underline{p}_{ra}^r \\ \phi \\ \omega \end{bmatrix}^{11\times 1} \tag{6.82}$$

The process filter models:

$$\begin{array}{ll} \dot{x}^a = v_x^a & \dot{p}_{x,ra}^r = 0 \\ \dot{y}^a = v_y^a & \dot{p}_{y,ra}^r = 0 \\ \dot{z}^a = v_z^a & \dot{p}_{z,ra}^r = 0 \\ \dot{v}_x^a = 0 & \dot{\Phi} = 0 \\ \dot{v}_y^a = -\omega^2 y^a & \dot{\omega} = 0 \\ \dot{v}_z^a = -\omega^2 z^a & \end{array} \tag{6.83}$$

The measurement model:

$$\begin{array}{l} \underline{z} \text{ is in cartesian coordinates} \\ \underline{z} = \underline{h}(\underline{x}) \\ \underline{h} = \underline{p}_{ra}^r + R_a^r(\Phi)\underline{p}^a \\ R \text{ is transformed from spherical to cartesian coordinates} \end{array} \tag{6.84}$$

6.4 Description of Solution

The Matlab programs are divided into 4 main files that can be seen in appendix C on page 103. The programs are summarized as follows:

1. Program estimating position and velocity of an airplane moving in a horizontal wave motion. The perturbations are unknown for the filters. This is solved by using a linear process filter model.

2. Program estimating position and velocity of an airplane moving in a helix motion. The Perturbations are unknown for the filters. This is solved by using a linear process filter model.

3. Program estimating position, velocity, $\underline{p}^r_{ra}$, Φ and ω of an airplane moving in a horizontal wave motion. The Perturbations are known for the filters. The process filter models are non-linear.

4. Program estimating position, velocity, $\underline{p}^r_{ra}$, Φ and ω of an airplane moving in a helix motion. The Perturbations are known for the filters. The process filter models are non-linear.

The filters used for estimation are EKF and SPF in all four cases.

EKF and SPF recives measurements from the radar in the radar frame $\mathcal{F}_r$, where the measurement noise for all the simulations are additive white noise drawn from a normal distribution

$$\underline{w}_k \sim N(0, R_k) \tag{6.85}$$

where R_k is for all k:

$$R = \begin{bmatrix} 50^2 & 0 & 0 \\ 0 & 0.001^2 & 0 \\ 0 & 0 & 0.001^2 \end{bmatrix} \tag{6.86}$$

where $\sigma_r = 50m$, $\sigma_a z = 0.001rad$ and $\sigma_e l = 0.001rad$. They are in spherical coordinates which for the case where the perturbations are known for the filter, are transformed to cartesian coordinates using equations 2.65 - 2.67. In addition the measurement covariance matrix R must be transformed to cartesian coordinates, this is done for every measurement updates, using the measurements in spherical coordinates from the radar as described in equations 2.68 - 2.73. The measurements are given in constant intervals of 1Hz. Both filters know the true noise variances in spherical coordinates and use the same measurements in each run. The time updates are in constant intervals of 100Hz.

The process noise for all the simulations are drawn from a normal distribution

$$\underline{v}_k \sim N(0, Q_k) \tag{6.87}$$

The choice of process noise are different for the two cases where the perturbations are known or unknown for the filters. This is because the process noise is a value describing how well you trust the filter model. Therefore it is used a larger value of the process noise in the case where the perturbations are unknown for the filter, giving greater respons to the measurements. Process covariance where the perturbations are known for the filters are set to:

$$Q = I\sigma_v^2 \tag{6.88}$$

where $\sigma_v = 0.05m/s$. The process covariance where the perturbations are unknown for the filters are:

$$Q = I\sigma_v^2 \tag{6.89}$$

where $\sigma_v = 0.95m/s$.

Initializations

First the initialization of the state vector for the case where the perturbations are unknown for the filters, are done by setting the initial position to the first measurement received from the radar. The velocities are initialized by calculating the average of the two first position measurements. This is done according to [22] as:

$$\underline{\hat{x}}_0 = \begin{bmatrix} x_0 & y_0 & z_0 & \frac{x_1 - x_0}{\Delta t} & \frac{y_1 - y_0}{\Delta t} & \frac{z_1 - z_0}{\Delta t} \end{bmatrix} \tag{6.90}$$

The covariance matrix is initialized using 2 Point Differentiation. This method is recommended in [22] because it guaranties consistensy of the filters. Consistency checks if the estimated states converges to the true value. The 2 Point Differentiation initialization of the covariance matrix is done by first calculating the known measurement covariance matrix from spherical to cartesian coordinates using equations 2.68 to 2.73 on page 13 and set the standard deviations on the diagonal of a 3×3 matrix. The initialization are then:

$$\sigma_{diag} = \begin{bmatrix} \sigma_x & 0 & 0 \\ 0 & \sigma_y & 0 \\ 0 & 0 & \sigma_z \end{bmatrix} \tag{6.91}$$

$$\hat{P}_0 = \begin{bmatrix} \sigma_{diag} & \frac{\sigma_{diag}}{\Delta t} \\ \frac{\sigma_{diag}}{\Delta t} & \frac{2\sigma_{diag}}{\Delta t} \end{bmatrix} \tag{6.92}$$

This is done equally for both EKF and SPF.

The initializations where the perturbations are known for the filters is a more challenging task. The main problem was to estimate position and velocity of the airplane using EKF and SPF. That is position and velocity in three directions, and an angular velocity of $\omega = 2\pi f$ in the horizontal and vertical plane. In addition the modell of the airplane is such that, it can be observed by the radar in every positions and heading angles between north and east. This means the initializations of the filters can not be done properly before measurements are received from the radar. Therefore it was created an initialization phase of the filters, where a roughly initialization of $\underline{\hat{x}}$ and a roughly initialization of the angle the aircraft are moving in with respect of the radar, namely the heading angle. MUSIC are used to initialize $\hat{\omega}$ by use of the estimated frequency. The heading angle are initialized by calculating the angle of the course, where the course are calculated from linear regression from the measurements, see figure 6.6. This angle depends on if the aircraft are moving towards the radar, or away from the radar. If the airplane is moving away from the radar from north to east, ϕ is given by:

$$\phi = \frac{3\pi}{2} + \arctan \frac{course_{y0} - course_{y1}}{course_{x1} - course_{x0}} \tag{6.93}$$

and towards the radar:

$$\phi = \pi + \arctan \frac{course_{y0} - course_{y1}}{course_{x0} - course_{x1}} \tag{6.94}$$

Where $course_{x0}$ and $course_{y0}$ are the first points in the stored course vector, and $course_{x1}$

and $course_{y1}$ are the last calculated point in the course vector.

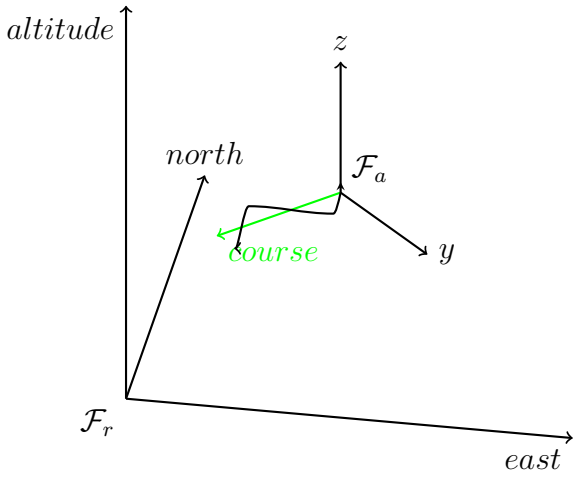

Figure 6.6: Radar frame, $\mathcal{F}_r$, and airplane frame, $\mathcal{F}_a$. Course along the x axis of $\mathcal{F}_a$ which is unknown for the filters. This is treated with linear regreassin of the measurements which gives an approximation of the course in green.

When Φ is calculated, the measurements can be rotated and set along the x-axis (east) in $\mathcal{F}_r$, and then initialize the velocities in $\mathcal{F}_a$ by the method described above. Position in $\mathcal{F}_a$ is initialized with $[0,0,0]^T$. $\underline{\hat{p}}^r_{ra}$ is initialized with the course vector. $\hat{\omega}$ is initialized as mention above, with the MUSIC estimate. This gives the state vector:

$$\underline{\hat{x}} = \begin{bmatrix} \underline{p}^a \\ \underline{v}^a \\ \underline{p}^r_{ra} \\ \phi \\ \omega \end{bmatrix} \tag{6.95}$$

Initialization of the covariance matrix where first done by 2 Point Differentiation. This was later redone by the author because the Monte Carlo simulations showed better results by setting the covariance of the measurement noise on the diagonal of $\hat{P}_0$.

The program flow for the helix motion with perturbations known for the filter are shown in algorithm 4.

Monte Carlo simulations and covariance analysis

The filters are compared to each other using Monte Carlo simulations. They are performed by generating different measurements each simulation affected by the measurement noise. The error trajectories were stored after each run. The mean value of the error, and the "real" covariance matrices, for both EKF ond SPF can be calculated as in [10] using the equations 6.96 and 6.97:

$$\underline{\hat{m}}^N_k = \frac{1}{N}\sum_{i=1}^{N} \underline{\hat{e}}^i_k \tag{6.96}$$

$$\hat{P}^N_k = \frac{1}{N-1}\sum_{i=1}^{N} (\underline{\hat{e}}^i_k - \underline{\hat{m}}^N_k)(\underline{\hat{e}}^i_k - \underline{\hat{m}}^N_k)^T \tag{6.97}$$

Algorithm 4 Program Flow

```
1: Initialize modell of helix
2: Initialize Monte Carlo simulations
3: for trajectory=1,2,...,number of MC runs do
4:     for t=1,2,... do
5:         Simulate airplane in F_a
6:         Rotate simulation of airplane in F_a to F_r
7:         Generate measurements with measurement noise from F_r
8:     end for
       {Initialize filter from measurements:}
9:     if Measeurement from radar then
10:        Transform measurement from spherical to cartesian coordinates
11:        Estimate frequancy from measurements using MUSIC
12:        Course calculation by linear regression
13:        Calculation of heading angle
14:        Initialization of p̂^r_ra set to last measurement
15:        Initialization of x̂^a
16:        Initialization of x̂^r
17:        Transform measurement covariance from spherical to cartesian coordinates
18:        Initialization of P̂_0
19:    end if
       {Start estimator:}
20:    for t=1,2,... do
21:        if Measurement update then
22:            Transform measurement from spherical to cartesian coordinates
23:            Estimate frequency from measurements using MUSIC
24:            Calculation of new measurement covariance, R
25:            Filter measurement update in F_a rotatet to F_r
26:        end if
27:        if Time update then
28:            Filter time update in F_a rotated to F_r
29:        end if
30:    end for{Stop estimator}
31:    Save error trajectories
32: end for
```

where N is the number of Monte Carlo runs and $\underline{\hat{e}}_k^i$ are the mean error trajectories for every time step k. The standard deviation are then:

$$\underline{\hat{s}}_k^N = \sqrt{\hat{P}_k^N} \tag{6.98}$$

All results are based on $N = 100$ Monte Carlo simulations.

6.5 Falling Body

Details of the mass falling toward the earth has been supplied by KDS. These details are reproduced here and the simulations of the system are given in figure 6.7.

6.5.1 Modelling of a falling body

A falling body is detected 61000 meters above a radar. The body has a true velocity of $3048\frac{m}{s}$ towards the earth. The falling body is exposed to gravity forces when entering the atmosphere and the differential equations for the falling body may be modeled as:

$$\dot{h} = -v \tag{6.99}$$

$$\dot{v} = \frac{\gamma e^{-\eta h} g v^2}{2\beta} + g \tag{6.100}$$

$$\dot{\beta} = 0 \tag{6.101}$$

where h is height and v is velocity. $g = 9.81\frac{m}{s}$ is the gravity constant, $\gamma = 1.754$, $\eta = 1.49 \times 10^{-4}$. $\beta = 19161\frac{kg}{ms^2}$ is the ballistic coefficient depending on mass, shape and cross sectional area.

The state space model can then be defined as:

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \end{bmatrix} = \begin{bmatrix} 0 & -1 & 0 \\ 0 & -\frac{\gamma e^{-\eta x_1} g x_2}{2x_3} & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} 0 \\ g \\ 0 \end{bmatrix} + \underline{v} \tag{6.102}$$

To simulate the process in Matlab the first order Euler method with discretization time $T = 0.1$ by equation 2.44 is applied. This gives the discrete model:

$$x_{1,k+1} = x_{1,k} - T x_{2,k} \tag{6.103}$$

$$x_{2,k+1} = x_{2,k} - \frac{T\gamma e^{-\eta x_{1,k}} g x_{2,k}^2}{2x_{3,k}} + Tg \tag{6.104}$$

$$x_{3,k+1} = x_{3,k} \tag{6.105}$$

The falling body is illustrated in figure 6.7.

6.5.2 The Filter Models

EKF model

Linearization of the process model equations 6.103 to 6.105 gives:

$$F = \begin{bmatrix} 0 & -1 & 0 \\ \frac{\gamma\eta e^{-\eta\hat{x}_1} g\hat{x}_2^2}{2\hat{x}_3} & -\frac{\gamma e^{-\eta\hat{x}_1} g\hat{x}_2}{\hat{x}_3} & \frac{\gamma e^{-\eta\hat{x}_1} g\hat{x}_2^2}{2\hat{x}_3^2} \\ 0 & 0 & 0 \end{bmatrix} \tag{6.106}$$

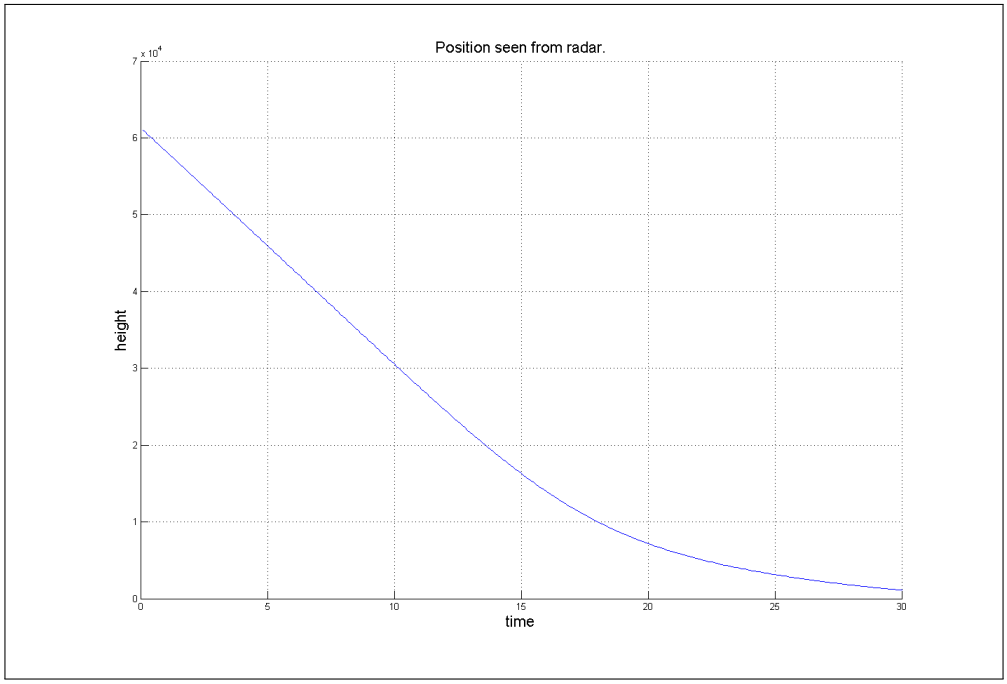

Figure 6.7: Falling body seen from the radar. Simulated for 30 seconds.

The discrete version of F is given by computing the transition matrix Φ:

$$\Phi = e^{FT} \tag{6.107}$$

The measurement Jacobian matrix H, is given straight forward as:

$$H = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix} \tag{6.108}$$

meaning that only position is measured. The Matlab code for EKF is given in appendix C.

SPF model

Here the measurement equation is linear, so the standard linear Kalman filter measurement update is used in the SPF. The sigma point vectors are propagated through the non-linear process model:

$$\bar{x}_{1,k+1} = \hat{x}_{1,k} - T\hat{x}_{1,k} \tag{6.109}$$

$$\bar{x}_{2,k+1} = \hat{x}_{2,k} - T\frac{\gamma e^{-\eta\hat{x}_{1,k}} g\hat{x}_{2,k}}{2\hat{x}_{3,k}} + Tg \tag{6.110}$$

$$\bar{x}_{3,k+1} = \hat{x}_{3,k} \tag{6.111}$$

The measurement matrix is given as in the EKF case equation 6.108. Matlab code for SPF is given in appendix C.

Chapter 7

Results

In each case the filters have been tested on three different scenarios:

1. Straigth against the radar from east.

2. Straight against the radar from north-east.

3. Away from the radar from north against east.

For each scenario there are shown figures of one single run where true trajectory for both position and velocity are compared with the filter estimates. From these figures it is not easy to say whether the performance are good or bad. Therefore it is in addition shown figures where the estimated mean errors are compared, and the filters standard deviations. This is performed by running 100 Monte Carlo simulations and for each runs the error trajectories are stored, and an average computed in the end. This gives a better picture of the filters performance. Both filters are using the same process noise covariance matrix Q and measurements. Below there are tables describing the abbreviations used in the figures legends.

One singel run, position and velocity.	
Abbreviation	Definition
true $\underline{p}^r$	The true trajectory of the airplane
true $\underline{v}_x^r$	The true velocity of the airplane in x direction
true $\underline{v}_y^r$	The true velocity of the airplane in y direction
true $\underline{v}_z^r$	The true velocity of the airplane in z direction
$\underline{z}$	Measurements from radar
10 sec.	A ten seconds interval
$\underline{\hat{p}}^r$ EKF	Estimated position for EKF
$\underline{\hat{p}}^r$ SPF	Estimated position for SPF
$\underline{\hat{v}}_x^r$	The estimated velocity of the airplane in x direction
$\underline{\hat{v}}_y^r$	The estimated velocity of the airplane in y direction
$\underline{\hat{v}}_z^r$	The estimated velocity of the airplane in z direction

Monte Carlo simulations.	
Abbreviation	Definition
SPF e	SPF mean estimation errors
EKF e	EKF mean estimation errors
S SPF	Standard deviations from SPF
S EKF	Standard deviations from EKF
S true SPF	True standard deviations from SPF
S true EKF	True standard deviations from EKF

All values are represented in the radar frame $\mathcal{F}_r$. For the figures showing position and velocity for one single run, the figures are read from right to left, except for scenario 3 where the airplane is moving away from the radar. The figures showing the Monte Carlo simulations with errors and standard deviations are read from left to right. All distance measurements are

7.1 Horizontal wave motion

7.1.1 Linear process filter model with non-linear measurement equation

Scenario 1.

Looking at figure 7.1 we can see the estimated position and velocities for both EKF and SPF for one single run. Both time update and measurement update are in the estimed positions. Therefore it is for every measurement update also a time update giving a corrector step and a prediction every second. The airplane is first located 18 kilometres east from the radar moving towards the radar in position $[0, 0, 0]$. True position/trajectory are the blue line, red and green lines are EKF and SPF estimates respectively. The black dots are the measurements and the purple circles are a 10 seconds interval. In the first four oscillations it seems that the EKF is slightly slower than SPF following the true position. But it looks like SPF and EKF performes almost equal as the airplane oscillates closer to the radar, and improves the estimates as time goes. Common for both filters, there are none good properties of prediction in the linear process filter model case. The same conclusions can be made looking at the velocities. Here, blue line is the true velocity in x direction and black line the true velocity in y direction. Red and green lines are estimated velocities for EKF and SPF.

Figure 7.2 shows the estimated position errors east (x), north (y) and altitude (z). Here it seems that EKF and SPF are performing almost equally for x and z estimations. Estimated position errors in y direction, EKF have some peeks that are higher than for SPF. The standard deviations are oscillating closer and closer to zero with no collapses.

In figure 7.3 we can see the estimated velocities for both EKF and SPF. The velocity in x direction, towards the radar, are very close to equal for both filters. In y direction SPF are performing better than EKF.

In [12] there have been performed a comparative study of non-linear filters for target tracking. The filters tested are among others EKF and SPF. It has been done with a linear process filter model on a linear process, and a 2D non-linear measurement matrix with range and bearing angle. Eight scenarios where tested between north and east, and the conclusions made in [12] emphasise the results given her. The SPF performed better than EKF in average.

The simulation results for scenarios 2 and 3 are shown in appendix B.

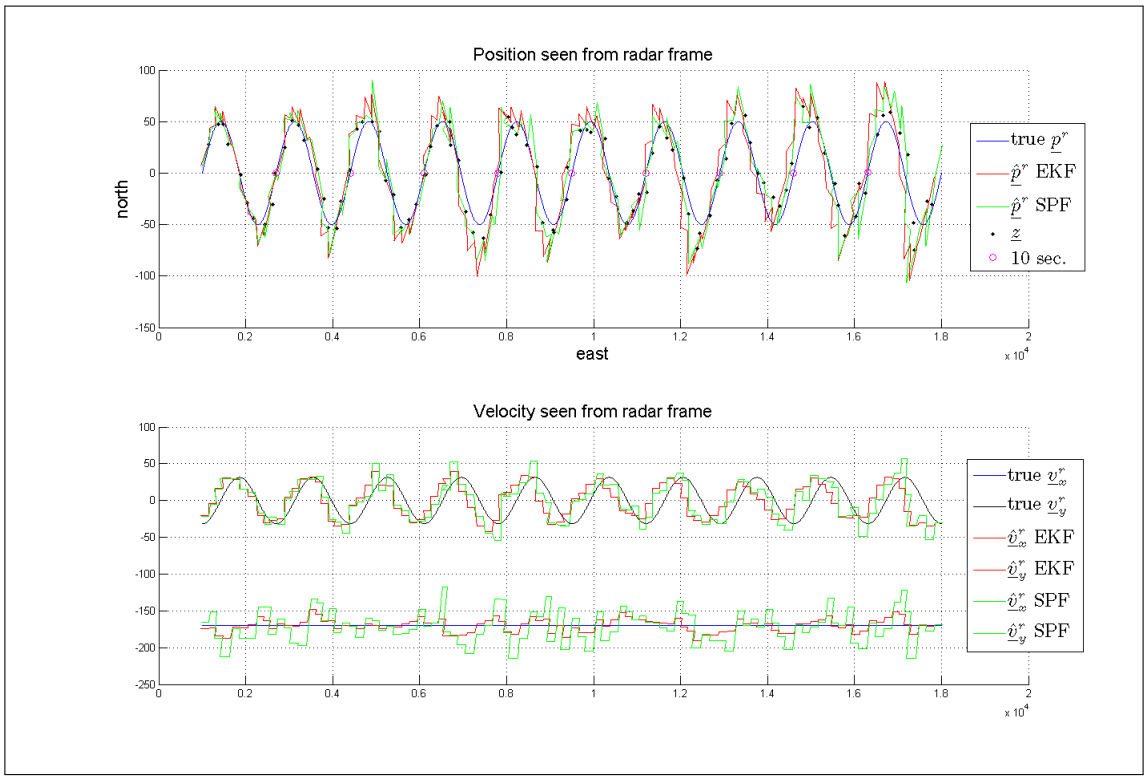

Figure 7.1: Scenario 1. Airplane position and velocities seen from $\mathcal{F}_r$ heading straight towards the radar from east. Perturbations unknown for the filters.

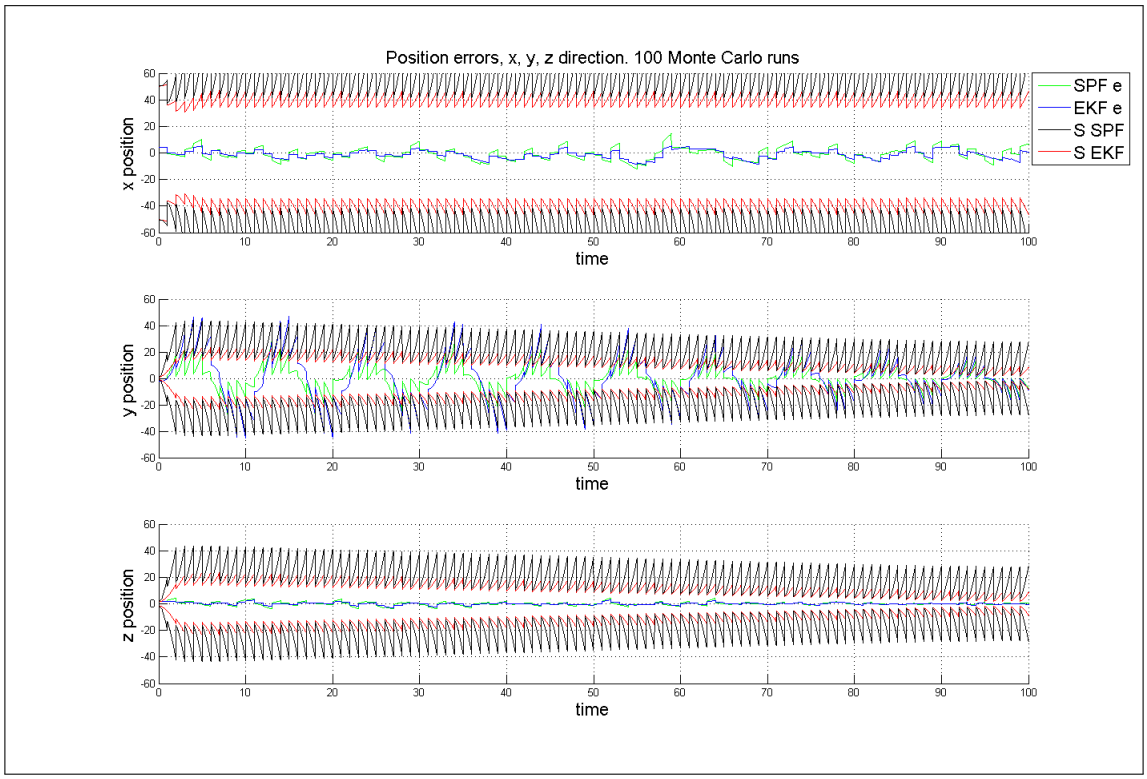

Figure 7.2: Scenario 1. Airplane position errors seen from $\mathcal{F}_r$ heading straight towards the radar from east. Perturbations unknown for the filters.

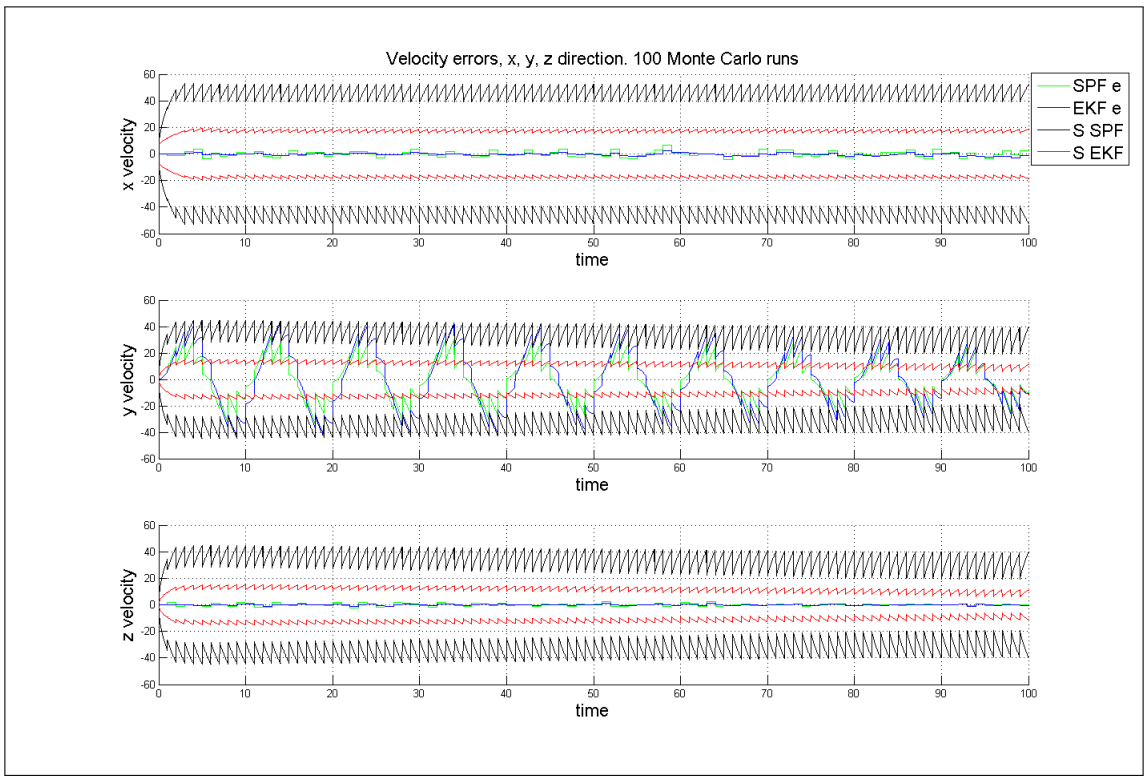

Figure 7.3: Scenario 1. Airplane velocity errors seen from $\mathcal{F}_r$ heading straight towards the radar from east. Perturbations unknown for the filters.

7.1.2 Non-linear process filter model

Scenario 1

In figure 7.4 there are figures of a typical simulation and estimation of the position and velocity for one single run for both EKF and SPF. The airplane are heading towards the radar from east. Here it is difficult to see which filter that performs better than the other, but for velocity, the SPF may perform slightly better than EKF for the velocity in north (y) direction.

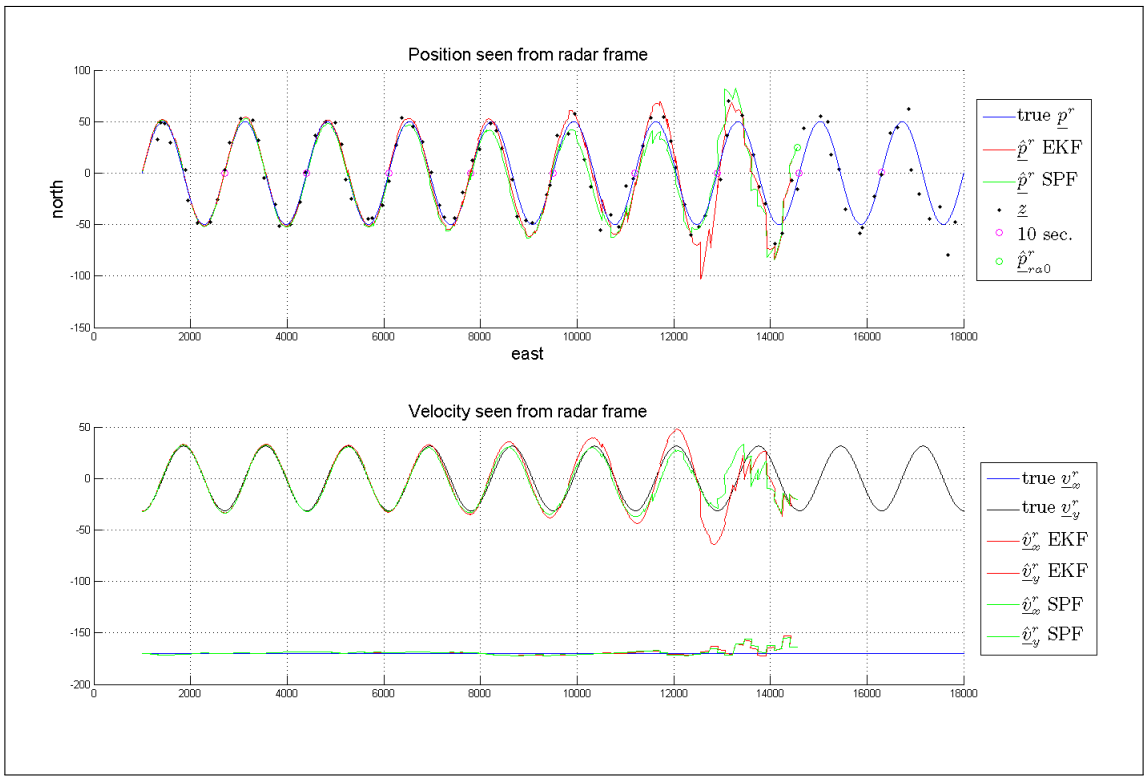

Figure 7.4: Scenario 1. Airplane position and velocity seen from $\mathcal{F}_r$. Perturbations known by the filters.

Figure 7.5, 7.6 and 7.7 shows position, velocity and $\underline{\hat{p}}^r_{ra}$ mean errors respectively. For the position errors the EKF performs better then SPF, especialy for north (y) direction. For both filters the standard deviations are decreasing. Looking at the velocities, SPF are performing better than EKF especialy for the velocity in north (y) direction. For east (x) and altitude (z), it is slightly any differences, both filters are performing satisfactory. The estimation of $\underline{\hat{p}}^r_{ra}$, the origo of $\mathcal{F}_a$ seen from $\mathcal{F}_r$, EKF performs better than SPF for these runs. SPF are drifting about 20 meters away from the true value in x-position. This can explain the performance of SPF mean position error in y direction in figure 7.5. Figure 7.8 shows the mean error of angular velocity $\hat{\omega}$ and the heading angle $\hat{\phi}$. $\hat{\omega}$ are estimated with music for both filter, which indicates that it vill give the same results. Estimated heading angle shows no differences in performance for the two filters. Figure 7.9 shows standard deviations produced by the filters, and true standard deviation. For position in x and z direction, the filter-produced standard deviations are 20 and 10 meters higher than the true standard deviation. This indicates that the filters belives they have a worse

estimate than they realy have. In y position the standard deviations are quite similar with the true covariance.

In [17] the second order EKF and SPF are compared for tracking maneuvering targets. The similarities with this scenario are an airplane moving in a horizontal sine wave in the x and y plane, along the x axis. The state vector used is $\begin{bmatrix} x & y & v_x & v_y \end{bmatrix}$. Here it was concluded that SPF made it more powerful than second order EKF tracking a sine wave along the x axis.

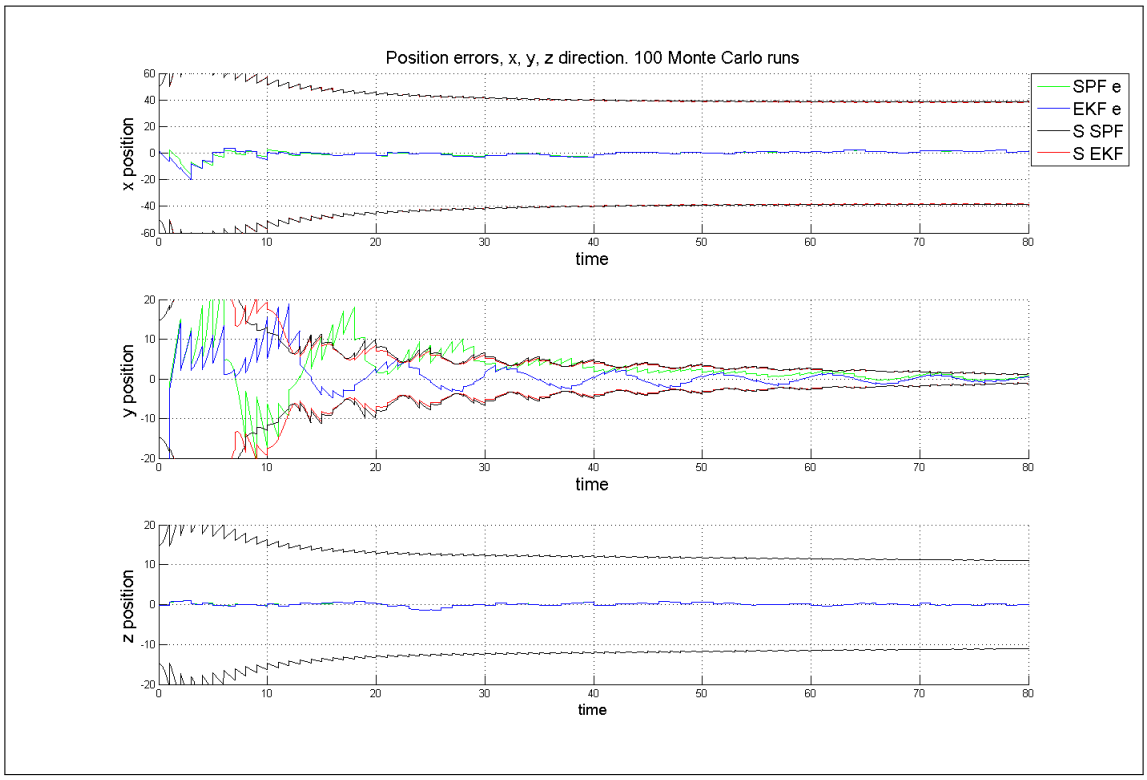

Figure 7.5: Scenario 1. Airplane position errors seen from $\mathcal{F}_r$. Perturbations known by the filters.

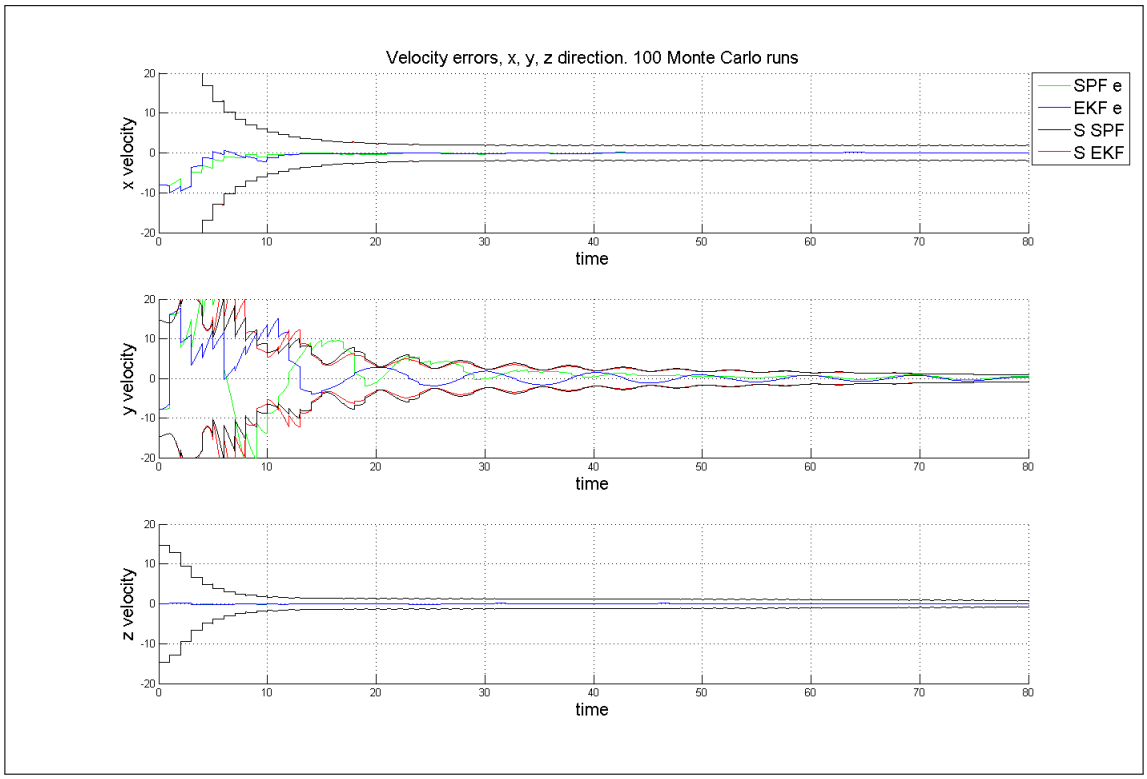

Figure 7.6: Scenario 1. Airplane velocity errors seen from $\mathcal{F}_r$. Perturbations known by the filters.

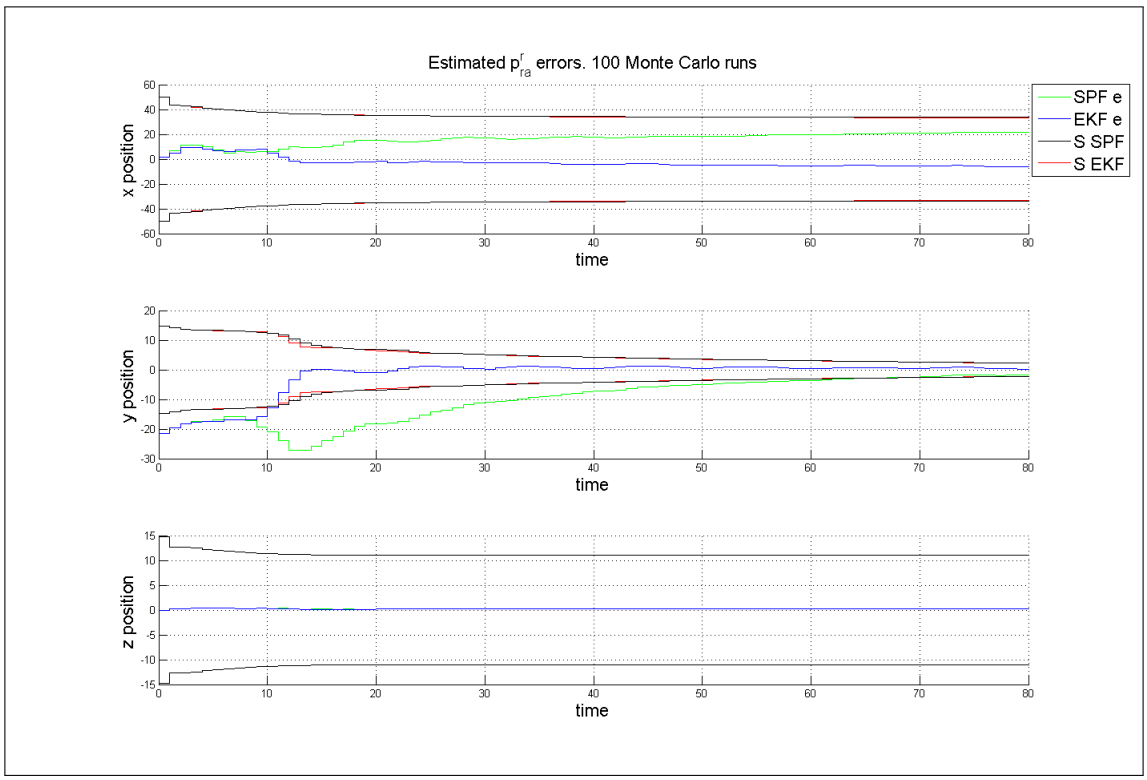

Figure 7.7: Scenario 1. Estimated $\underline{p}^r_{ra}$ errors seen from $\mathcal{F}_r$. Perturbations known by the filters.

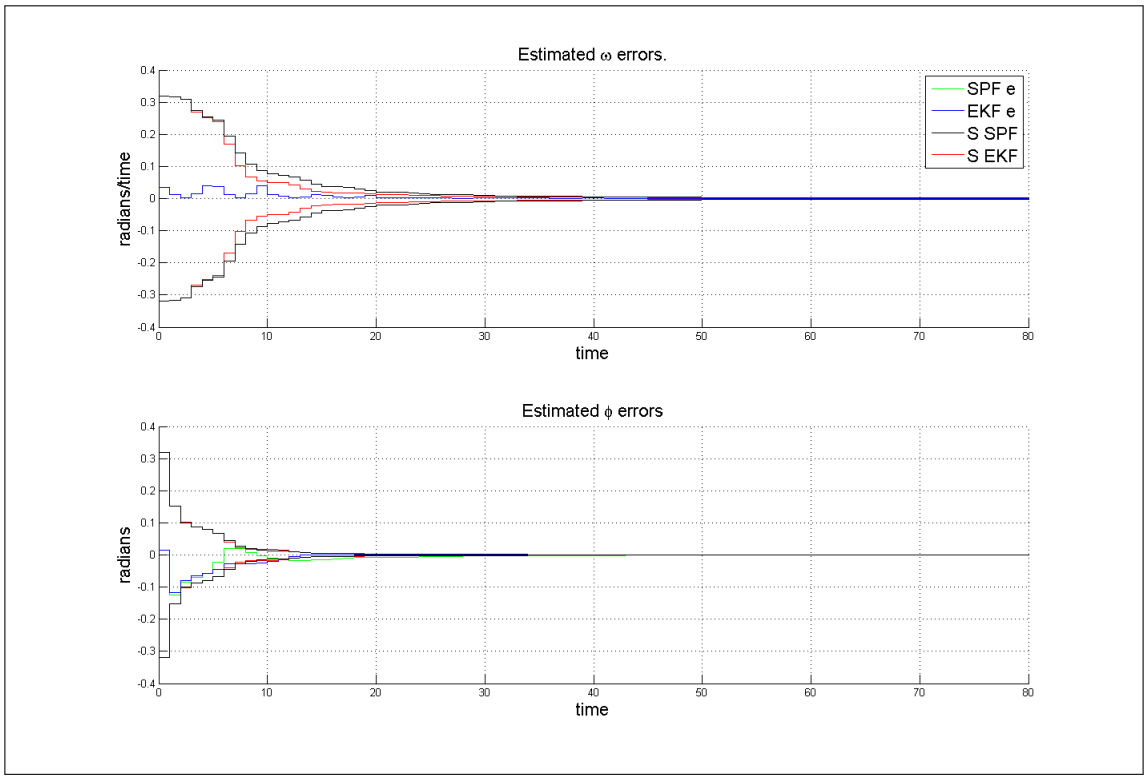

Figure 7.8: Scenario 1. Estimated ω and Φ errors. Perturbations known by the filters.

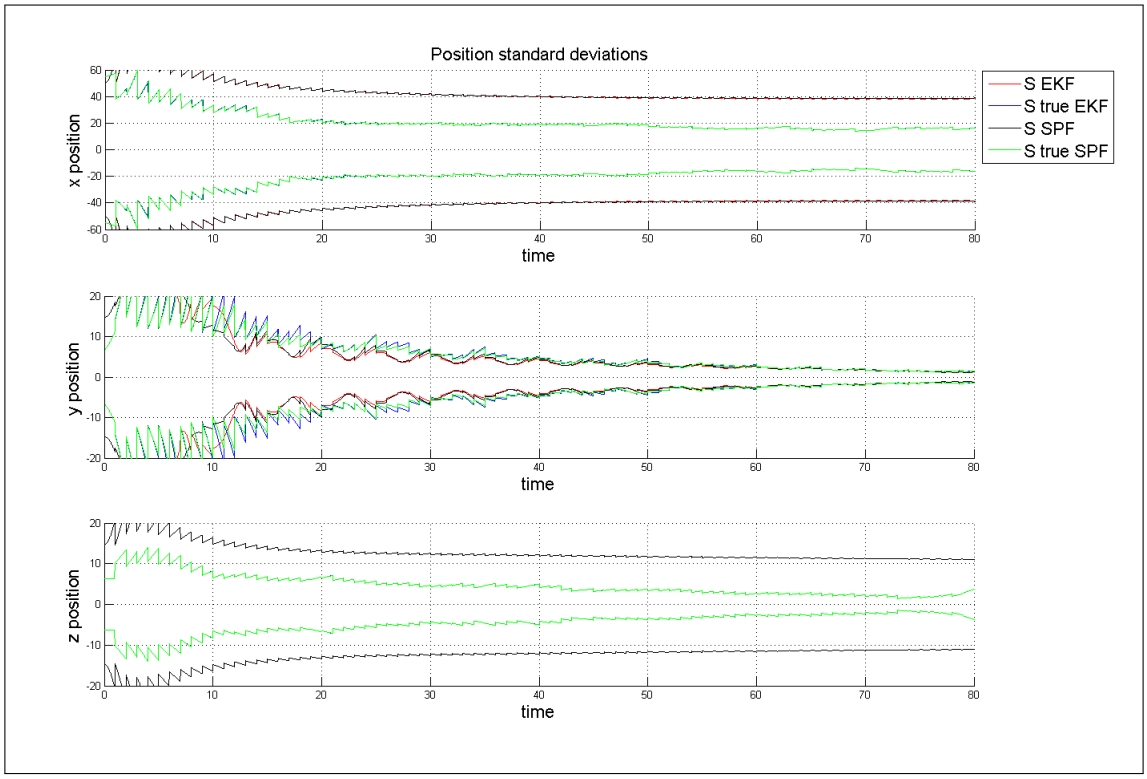

Figure 7.9: Scenario 1. Standard deviations from filters and true standard deviations.

Scenario 2

In figure 7.10 there are figures of a typical simulation and estimation of the position and velocity for one single run for both EKF and SPF where the airplane heading angle are 45 degrees above east. Here the velocity in $\mathcal{F}_r$ oscillates for both directions x and y. The velocity in x direction are linear in $\mathcal{F}_a$ and non-linear in $\mathcal{F}_r$. Since the filters estimates velocity in $\mathcal{F}_a$ and rotates to $\mathcal{F}_r$, the velocity estimates should not be any worse than for scenario 1, if the estimates of $\underline{p}^r_{ra}$ and heading angle ϕ are accurate. These two factors are implemented in the measurement equations for the filters, which should indicate accurate estimates.

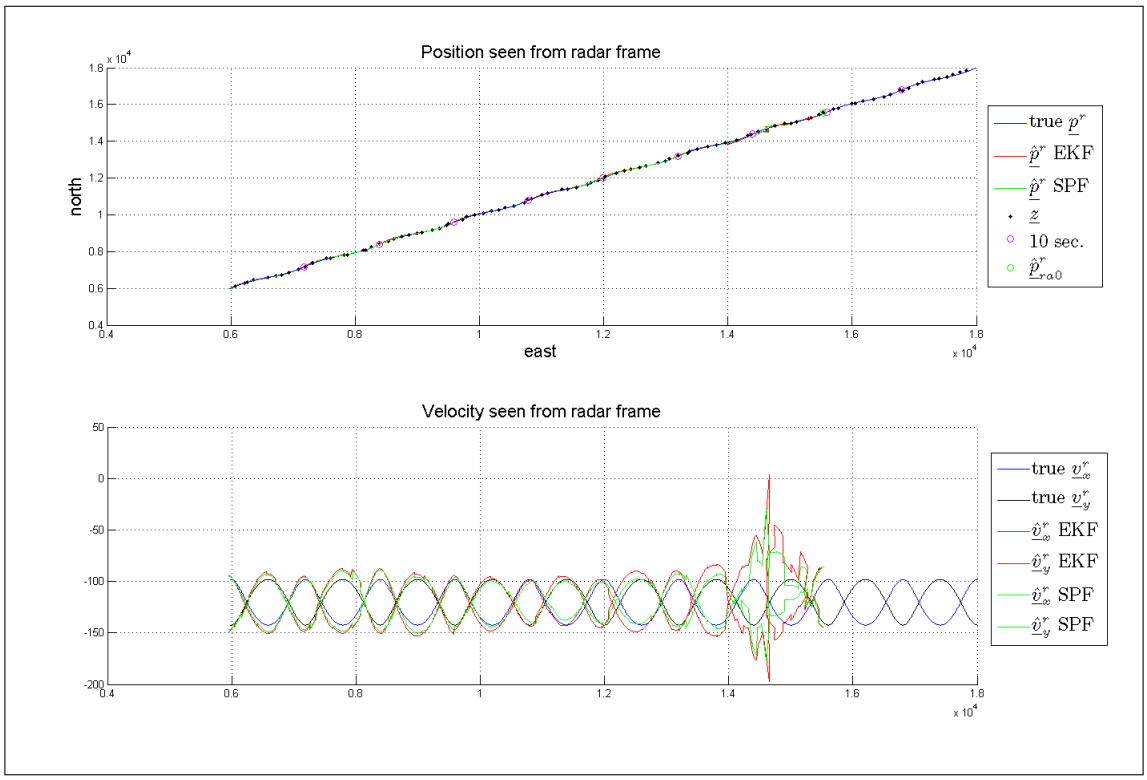

Figure 7.10: Scenario 2. Airplane position and velocity seen from $\mathcal{F}_r$. Perturbations known by the filters.

Figure 7.11, 7.12 shows estimated position and velocity errors respectively. These shows some decrease in performance from scenario 1. Estimated x position for SPF are oscillating slightly, but still it performes better than EKF. EKF has a mean error of about 10 meters versus about 5 to 10 meters for SPF. Mean errors for y position have increased with about 5 meters from scenario 1. Estimated z position remains the same as in scenario 1, which is as suspected since the airplane has a constant altitude. The velocity mean errors reflects the position errors, with increased errors in x direction and y direction. Still SPF performes better than EKF. Also in this scenario both filters have some problems estiamting $\underline{p}^r_{ra}$. In x position SPF are performing better than EKF with about 10 meters. In y position EKF is performing beetter than SPF. Here both filters are converging to a value 10 to 20 meters lower than the true value. Estimated ω and Φ shows no decrease in performance in figure 7.14 from scenario 1.

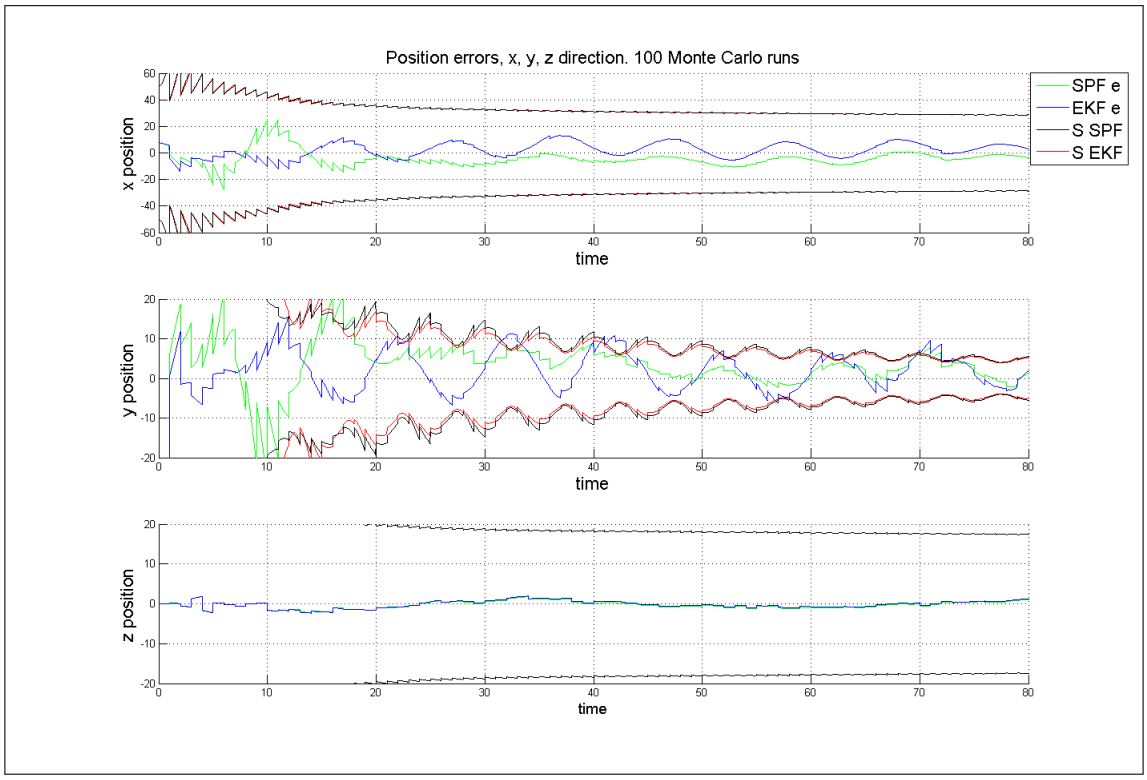

Figure 7.11: Scenario 2. Airplane position errors seen from $\mathcal{F}_r$. Perturbations known by the filters.

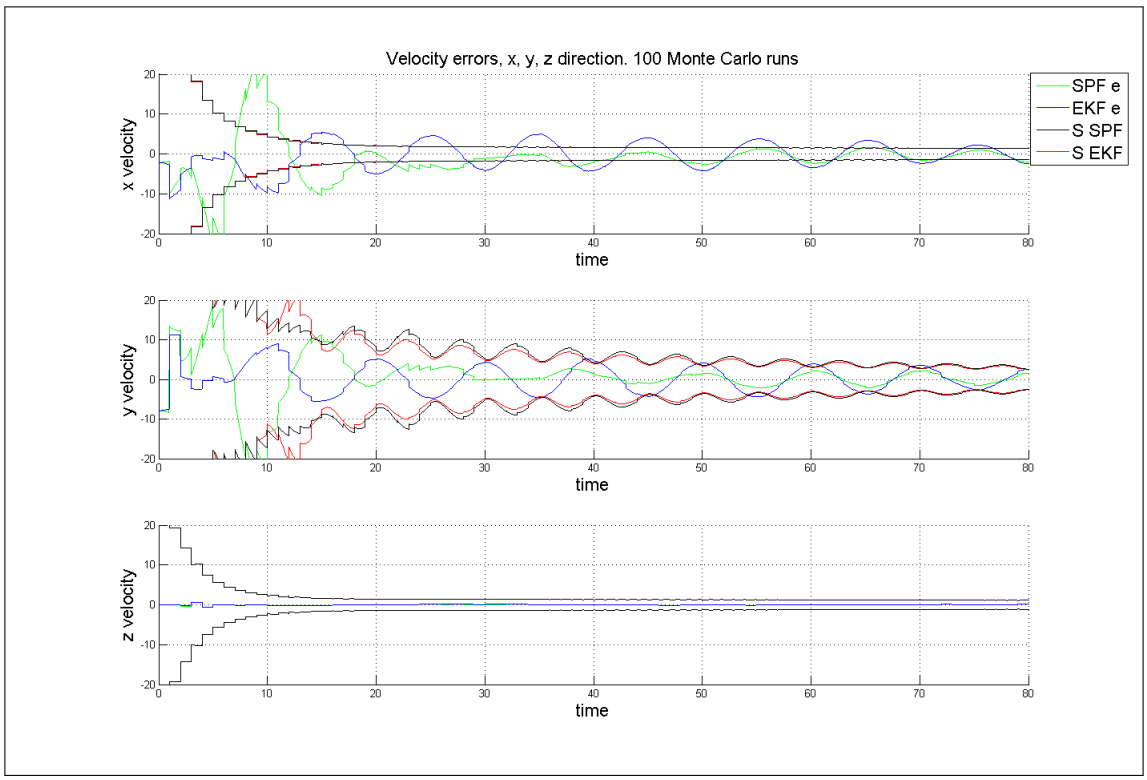

Figure 7.12: Scenario 2. Airplane velocity errors seen from $\mathcal{F}_r$. Perturbations known by the filters.

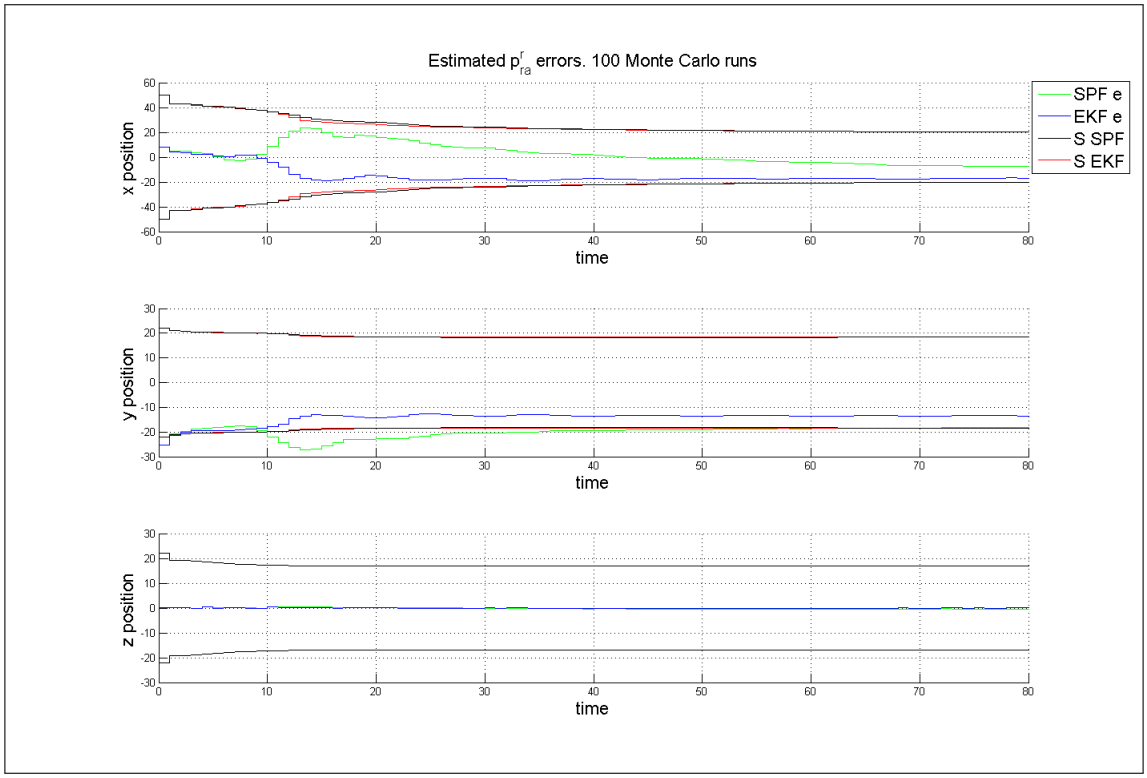

Figure 7.13: Scenario 2. Estimated $\underline{p}^r_{ra}$ errors seen from $\mathcal{F}_r$. Perturbations known by the filters.

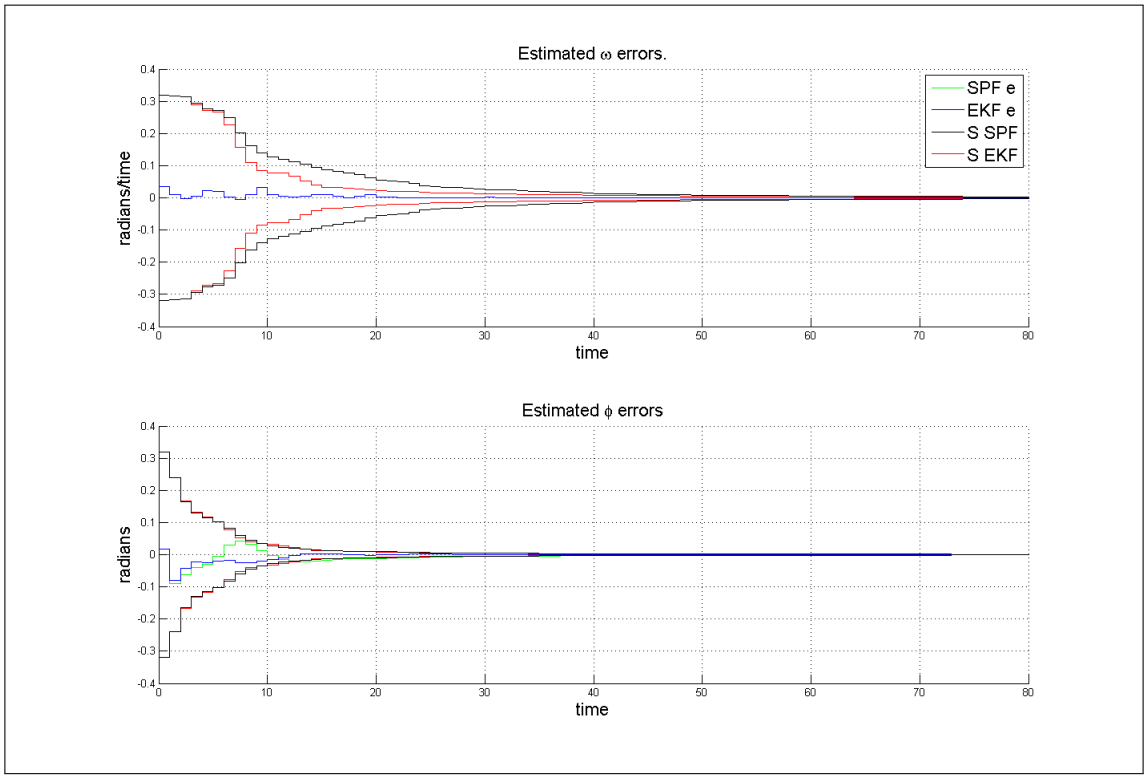

Figure 7.14: Scenario 2. Estimated ω and Φ errors. Perturbations known by the filters.

Scenario 3

In figure 7.15 there are figures of a typical simulation and estimation of the position and velocity for one single run for both EKF and SPF where the airplane is heading from north to east, away from the radar in $[0, 0, 0]$. Looking at the velocities in figure 7.15, it seems that SPF is performing better, particularly in the first 2000 meters. Figure 7.16, 7.17, 7.18 and 7.19 shows position, velocity, $\hat{\underline{p}}^r_{ra}$, $\hat{\omega}$ and $\hat{\Phi}$ errors respectively. These shows no decrease in performance from scenario 2. This indicates that the the airplane can come in radar seight from any positions and still give satisfactory results. As a rule of thumb there have been said that if $\frac{2}{3}$ of the mean errors are inside of the standard deviations, the estimates are good. The author have no text to refer to, but this is the case of the estimates for SPF in every scenarios. EKF have problems staying inside of the standard deviations for x velocity in scenario two and three.

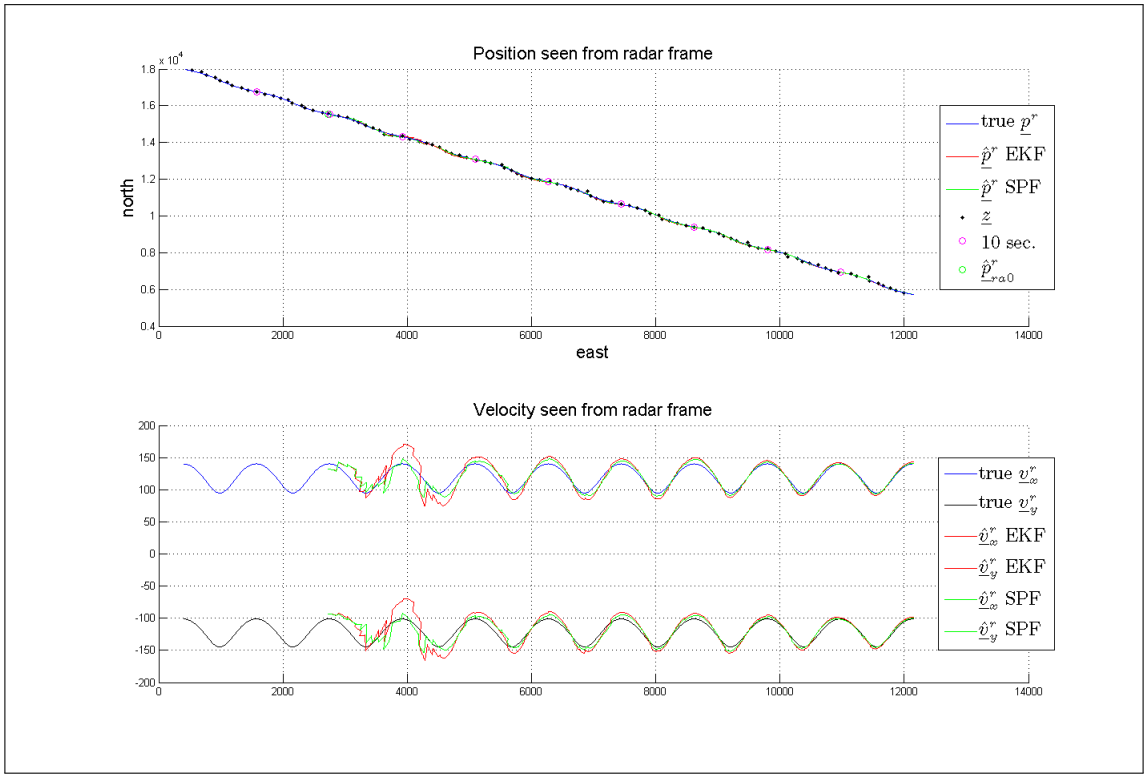

Figure 7.15: Scenario 3. Airplane position and velocity seen from $\mathcal{F}_r$. Perturbations known by the filters.

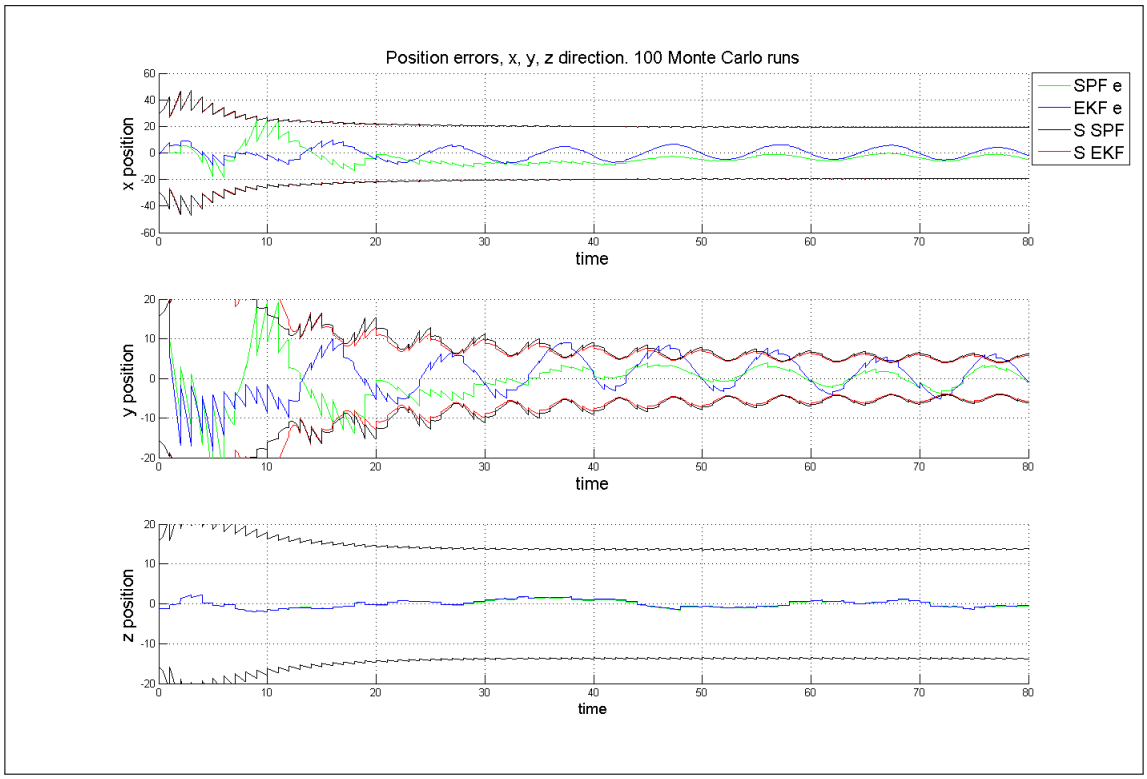

Figure 7.16: Scenario 3. Airplane position errors seen from $\mathcal{F}_r$. Perturbations known by the filters.

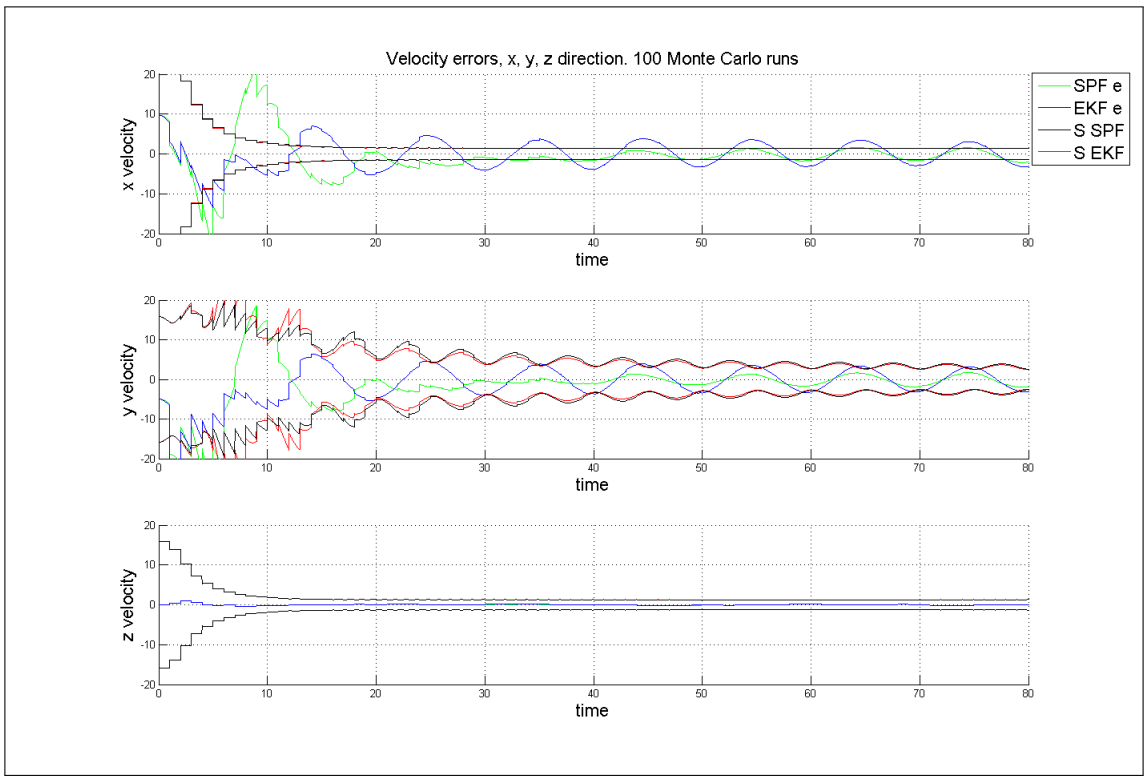

Figure 7.17: Scenario 3. Airplane velocity errors seen from $\mathcal{F}_r$. Perturbations known by the filters.

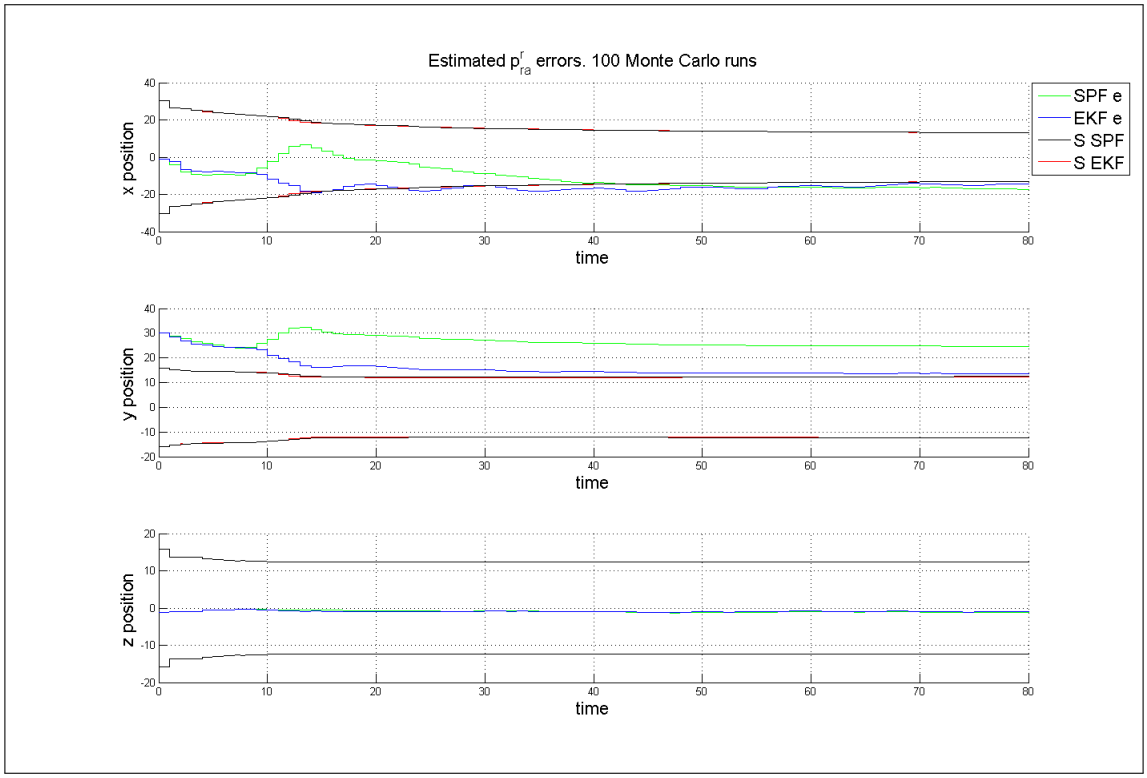

Figure 7.18: Scenario 3. Estimated $\underline{p}^r_{ra}$ errors seen from $\mathcal{F}_r$. Perturbations known by the filters.

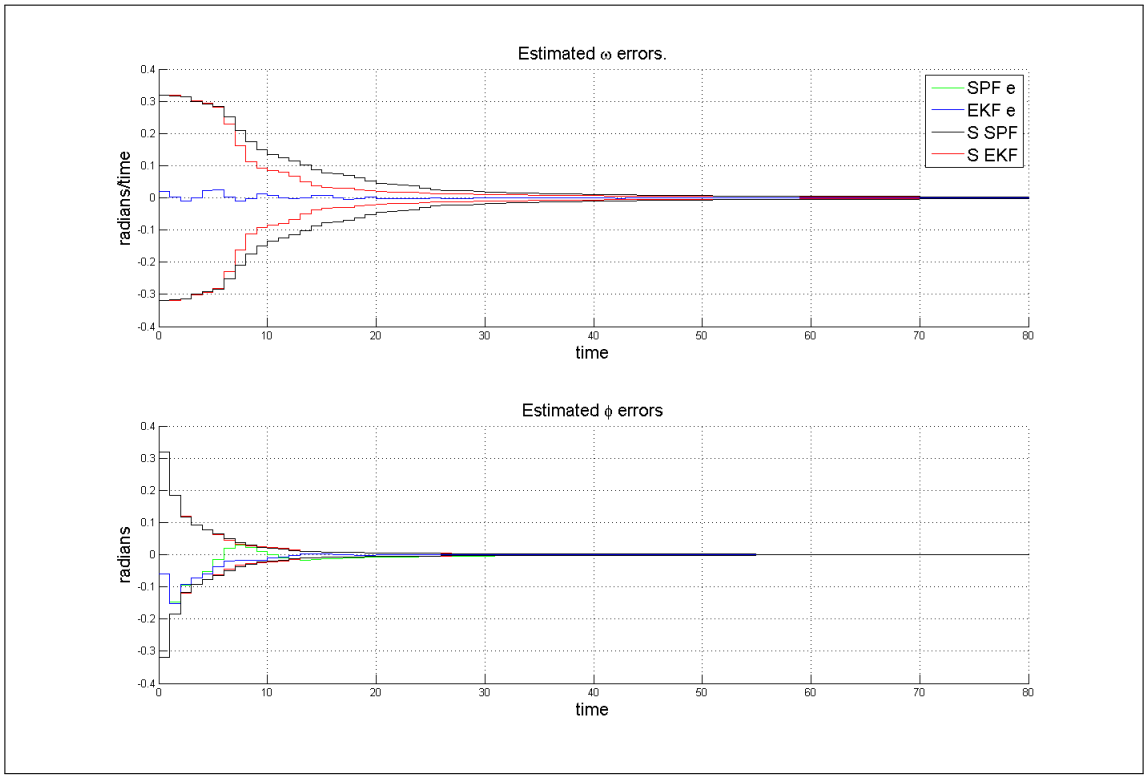

Figure 7.19: Scenario 2. Estimated ω and Φ errors. Perturbations known by the filters.

7.2 Helix motion

In addition to the horizontal wave motion, it is implemented a cosine perturbation in the vertical plane. This gives a more challenging scenario perturbing the trajectory in the horizontal plane and the vertical plane.

7.2.1 Linear process filter model with non-linear measurements

Scenario 1.

In figure 7.20 there are shown a typical simulation with SPF and EKF for position and velocity. The airplane are heading straight against the radar from east. The figure can not tell if one filter performes better than the other. Like in the horizontal wave motion case, we can see that both filters have no good prediction properties when the filter model is linear, and the true trajectory is non-linear. Figure 7.21 and 7.22 shows position and velocity mean errors. The performanc of SPF are again better than EKF. SPF have lower peaks than EKF, and in position errors in x direction, EKF have some divergence after 90 seconds.

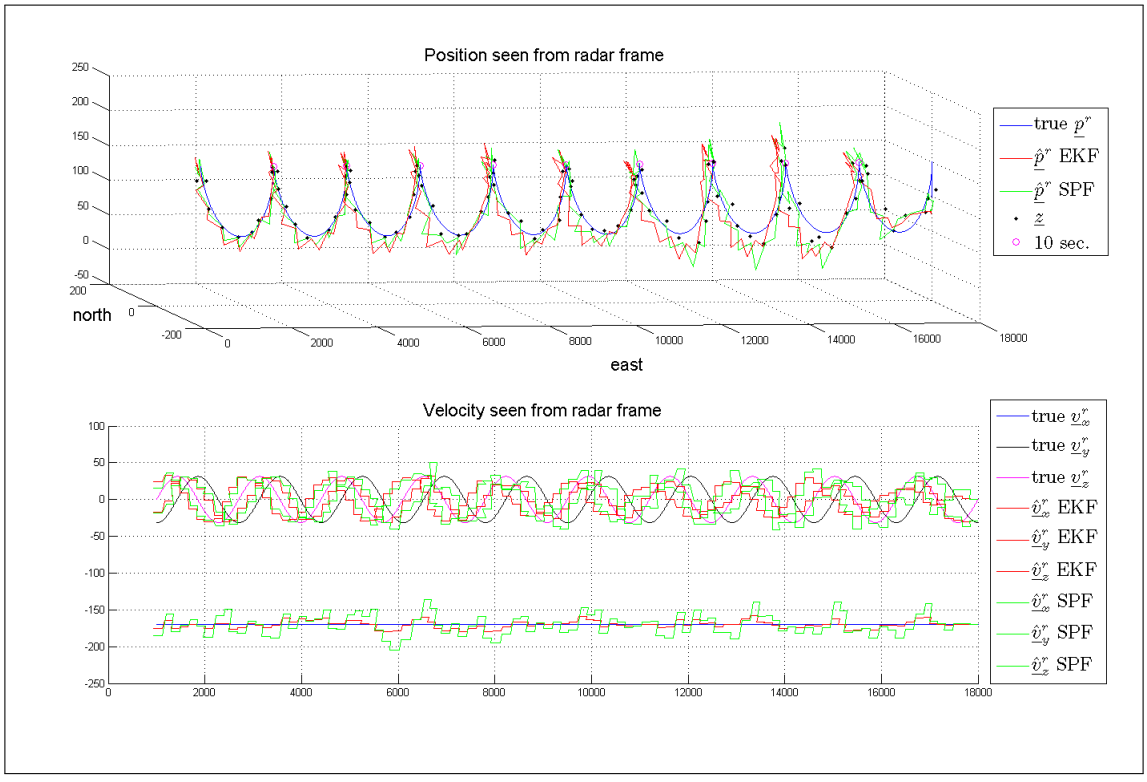

Figure 7.20: Scenario 1. Airplane position and velocities seen from $\mathcal{F}_r$ heading towards the radar east. Perturbations unknown for the filters.

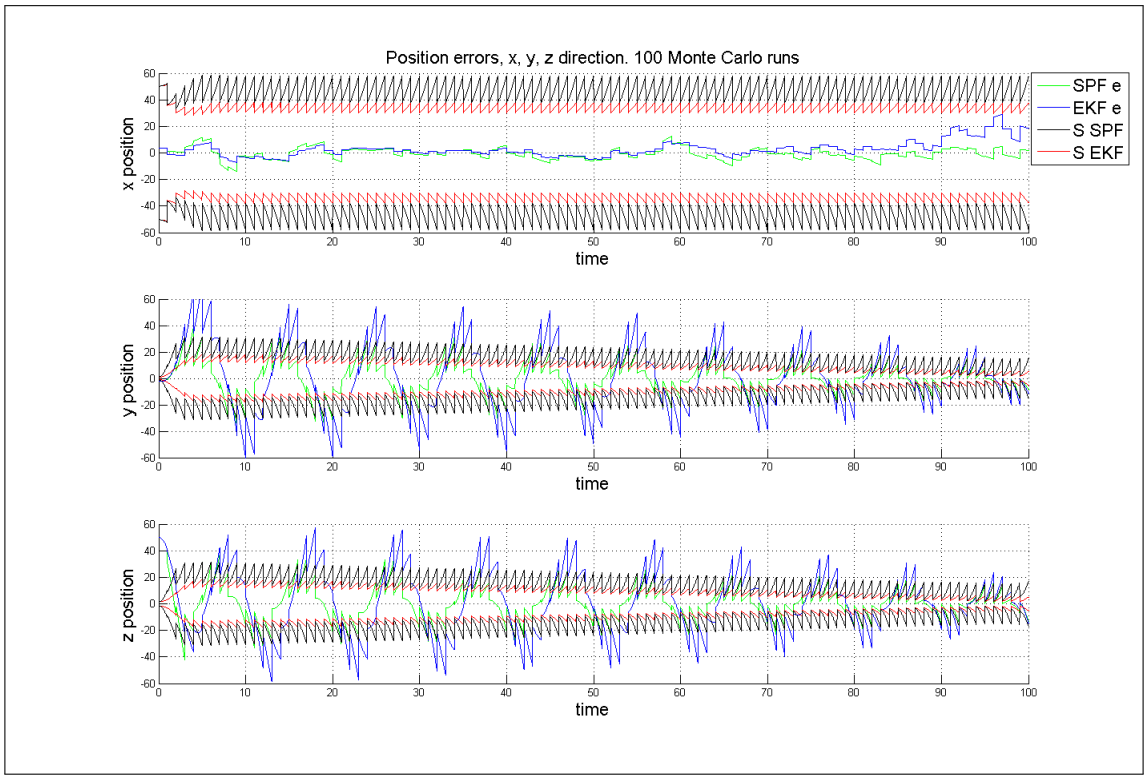

Figure 7.21: Scenario 1. Airplane position errors seen from $\mathcal{F}_r$ heading towards the radar east. Perturbations unknown for the filters.

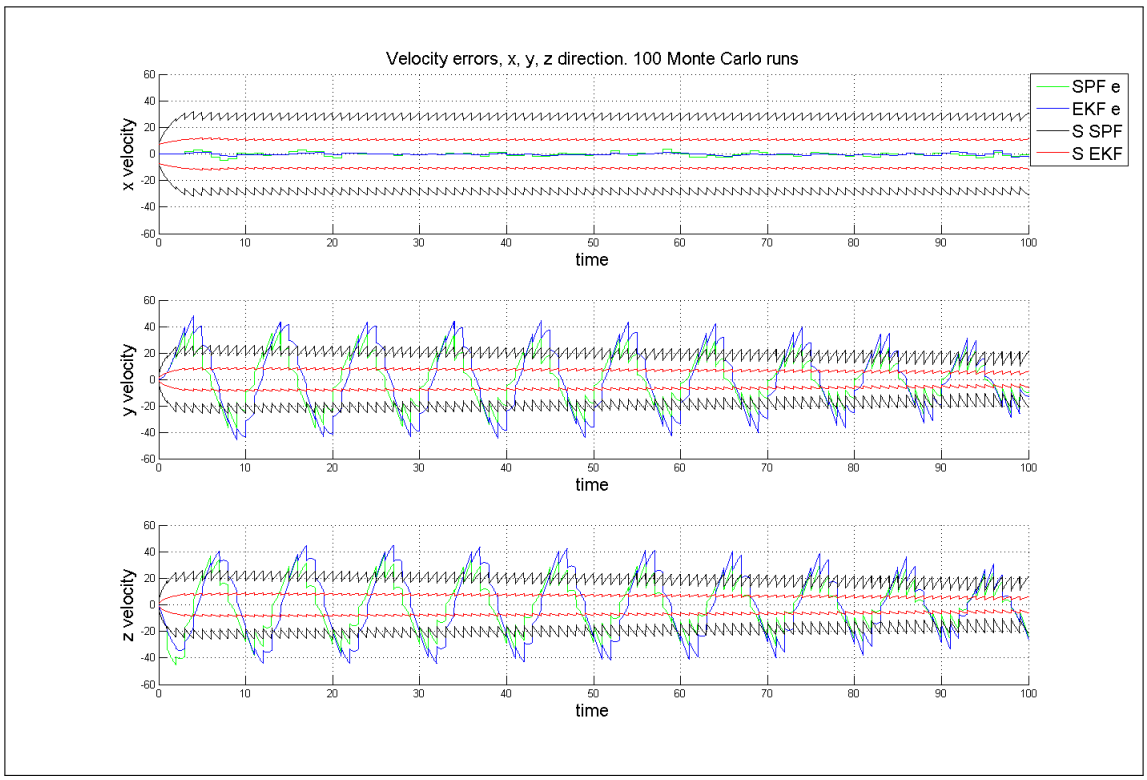

Figure 7.22: Scenario 1. Airplane velocity errors seen from $\mathcal{F}_r$ heading towards the radar east. Perturbations unknown for the filters.

7.2.2 Non-linear process filter model

Scenario 1

In figure 7.23 there are figures of a typical simulation and estimation of the position and velocity for one single run for both EKF and SPF. Here we ccan see much more acccurate prediction proporties of both filters. Figure 7.24, 7.25 and 7.26 shows position, velocity and $\hat{\underline{p}}^r_{ra}$ errors respectively. For position error in east (x) direction both filters are performing quite equally. For north (y) and altitude (z) directions the Sigma Point filter is slightly better than EKF. In figure 7.26 we can see that both filters are having problems estimatein $\underline{p}^r_{ra}$, like we saw in the horizontal wave motion case. In y position EKF performes great in comparison with SPF. x position is estimated slightly better by SPF. In figure 7.27 $\hat{\Phi}$ and $\hat{\omega}$ mean errors are shown. These errors are also in the helix case very low for both filters. Figure 7.28 shows the filters standard deviations compared with the true standard deviations. It can be summarized as in the horizontal wave motion case. For position in x direction the filters standard deviations are 20 meters higher than the true standard deviation. This indicates that the filters belive they have a worse estimate than they really have. In y and z position the filters standard deviations are close to the true standard deviations. But the true standard deviations for z direction drifts close to the end of the simulation.

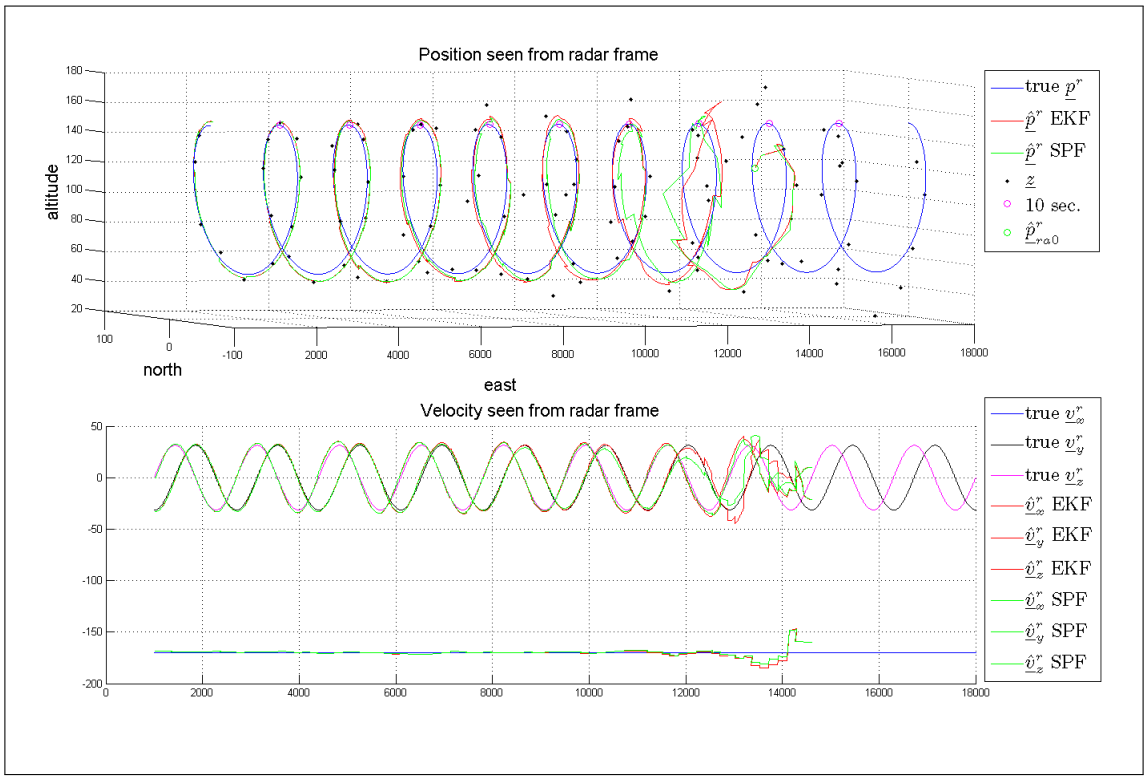

Figure 7.23: Scenario 1. Airplane position and velocity seen from $\mathcal{F}_r$. Perturbations known by the filters.

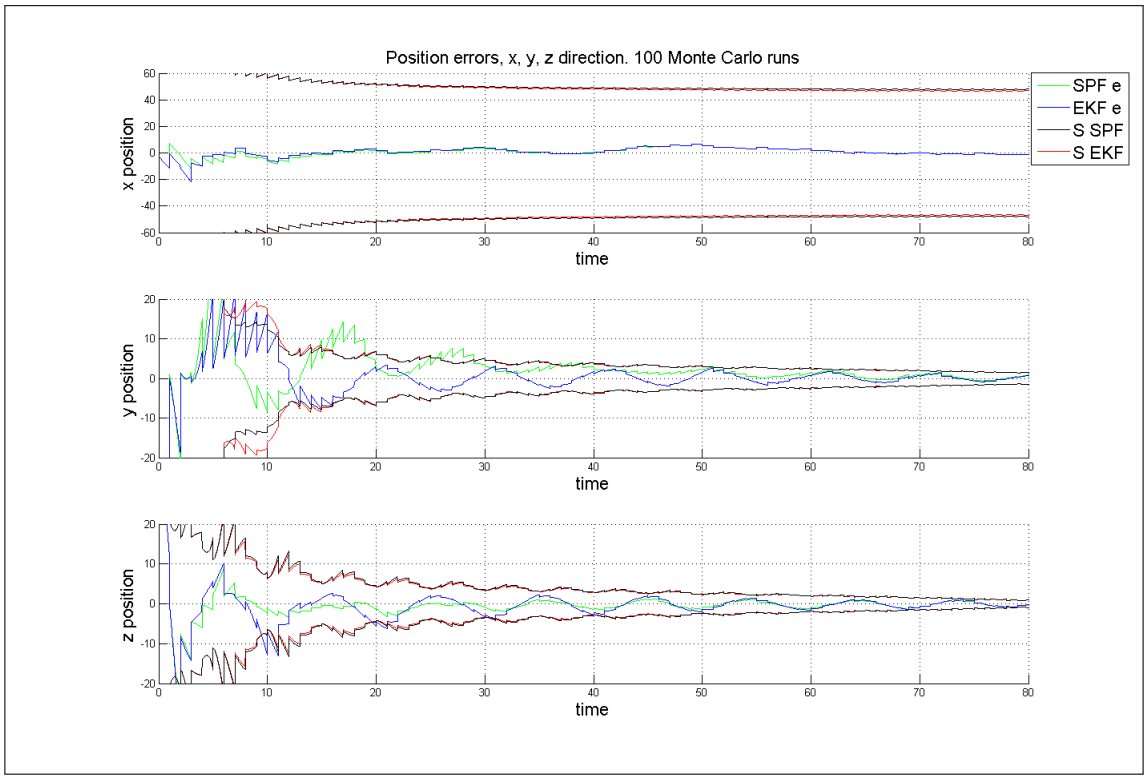

Figure 7.24: Scenario 1. Airplane position errors seen from $\mathcal{F}_r$. Perturbations known by the filters.

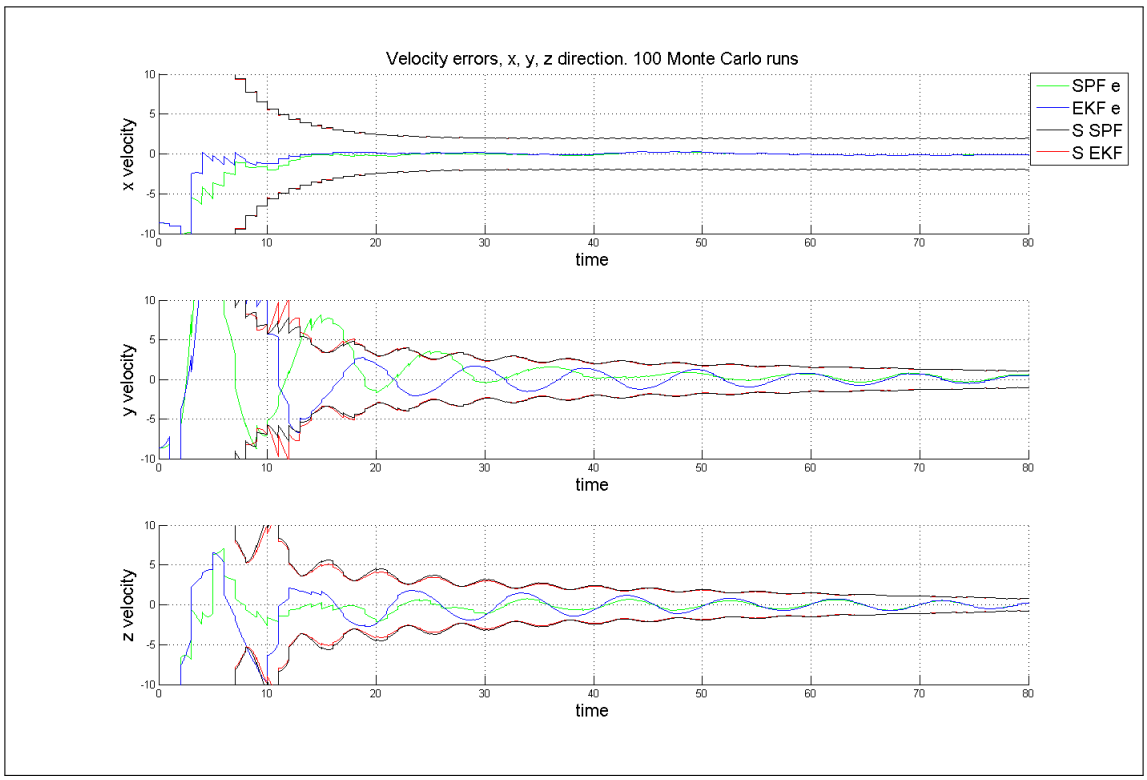

Figure 7.25: Scenario 1. Airplane velocity errors seen from $\mathcal{F}_r$. Perturbations known by the filters.

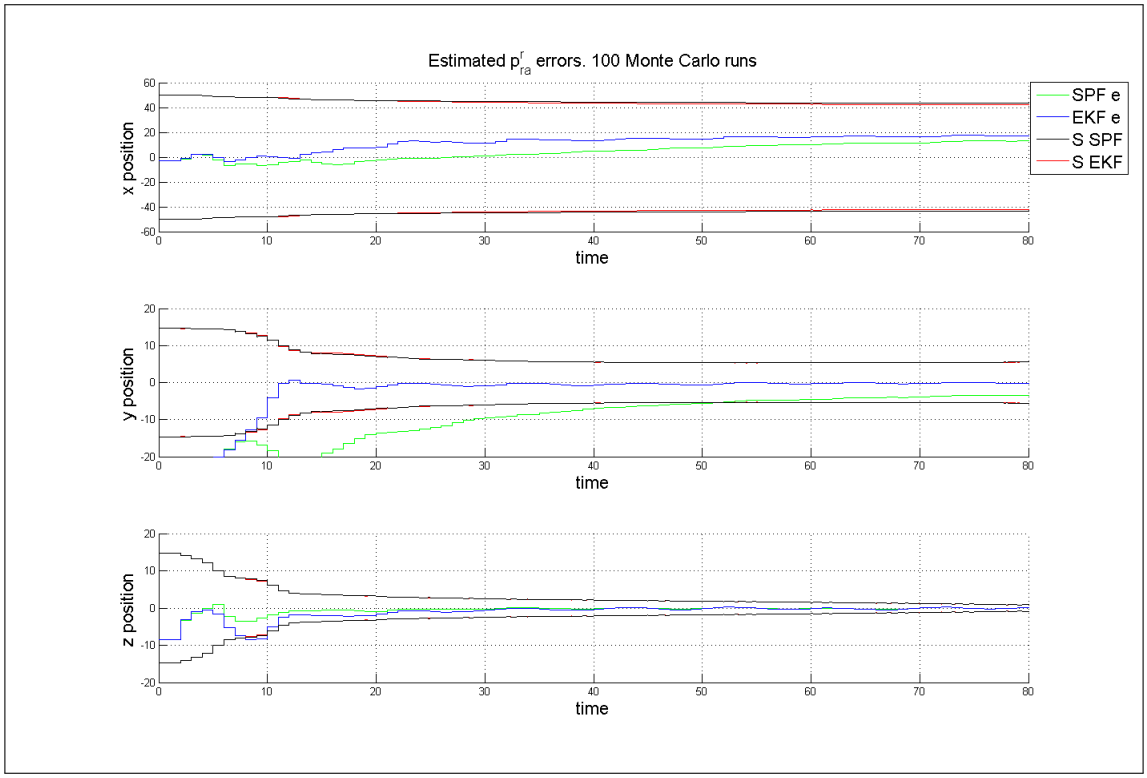

Figure 7.26: Scenario 1. Estimated $\underline{p}^r_{ra}$ errors seen from $\mathcal{F}_r$. Perturbations known by the filters.

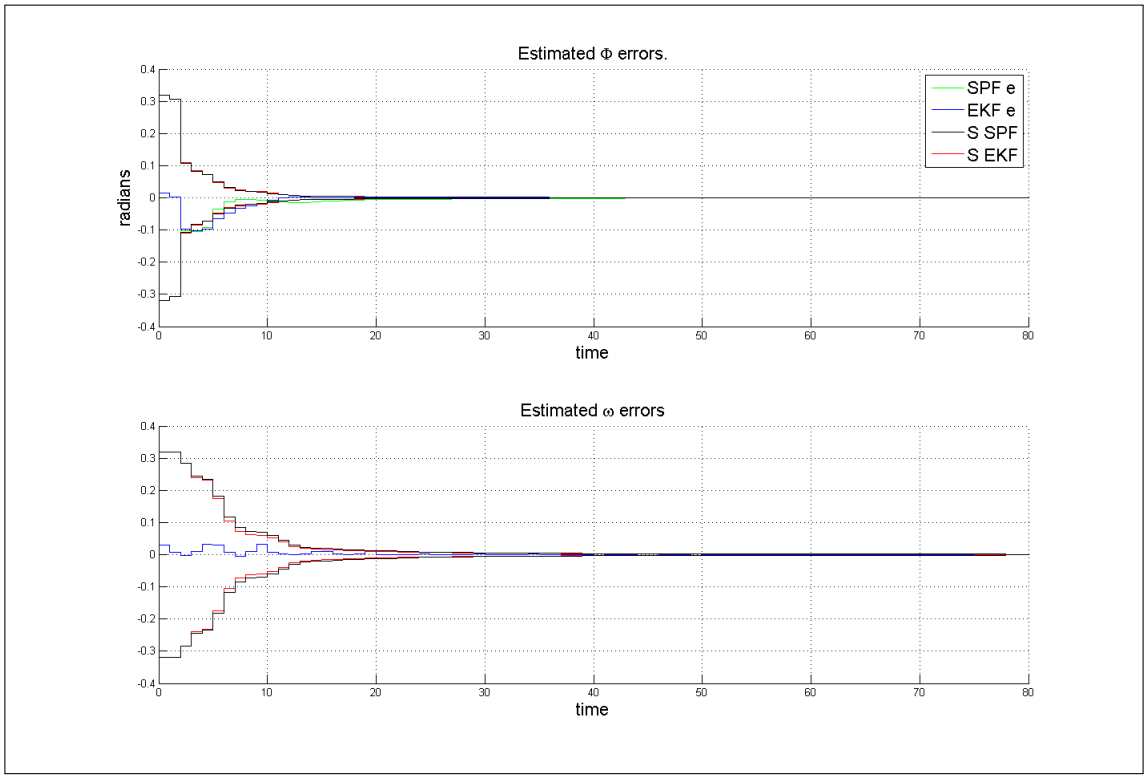

Figure 7.27: Scenario 1. Estimated ω and Φ errors. Perturbations known by the filters.

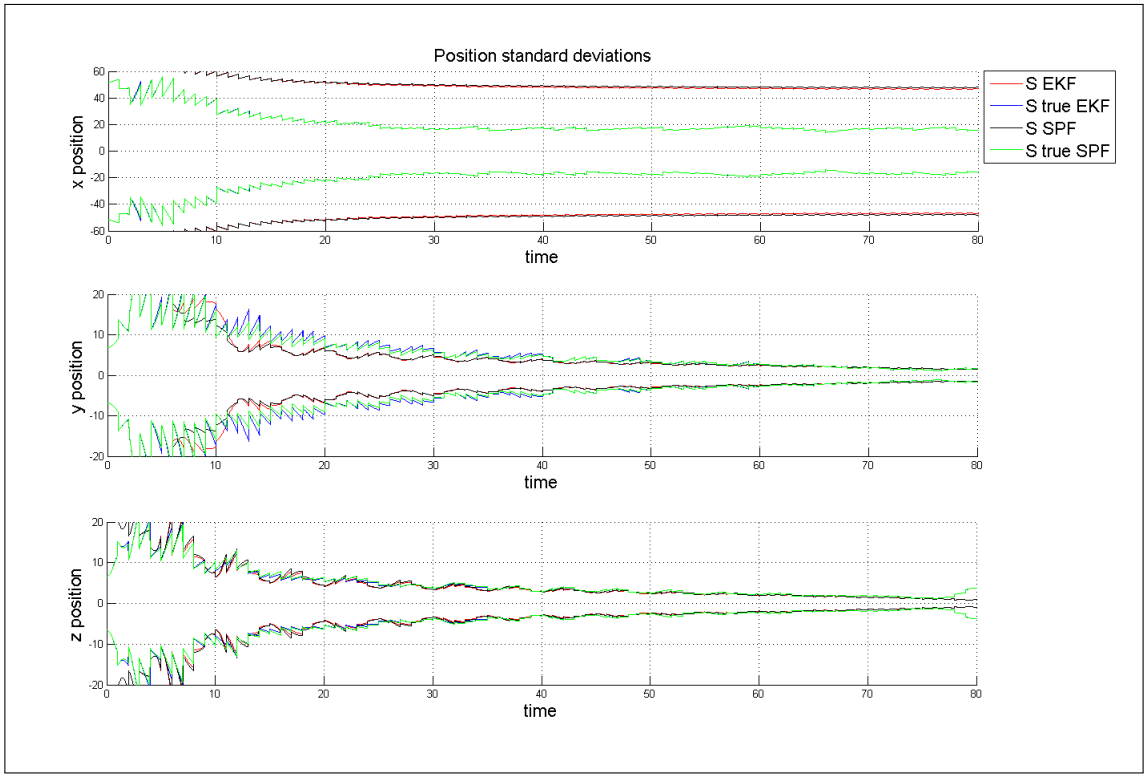

Figure 7.28: Scenario 1. Standard deviations from filters and true standard deviations.

Scenario 2

Figure 7.29 shows an airplane heading straight against the radar from north east. In figures 7.30 and 7.31 shows position and velocity mean errors. This shows the same reuslts as in the horizontal wave motion case. The filters have more problems estimating x position and x velocity. But all over the estimates are inside of the standard deviations two thirds of the time which indicates good estimates by the rule of thumb. Also in this scenario the $\hat{\underline{p}}^r_{ra}$ mean errors in figure 7.32 are not satisfactory. Estimated Φ and ω still gives great results.

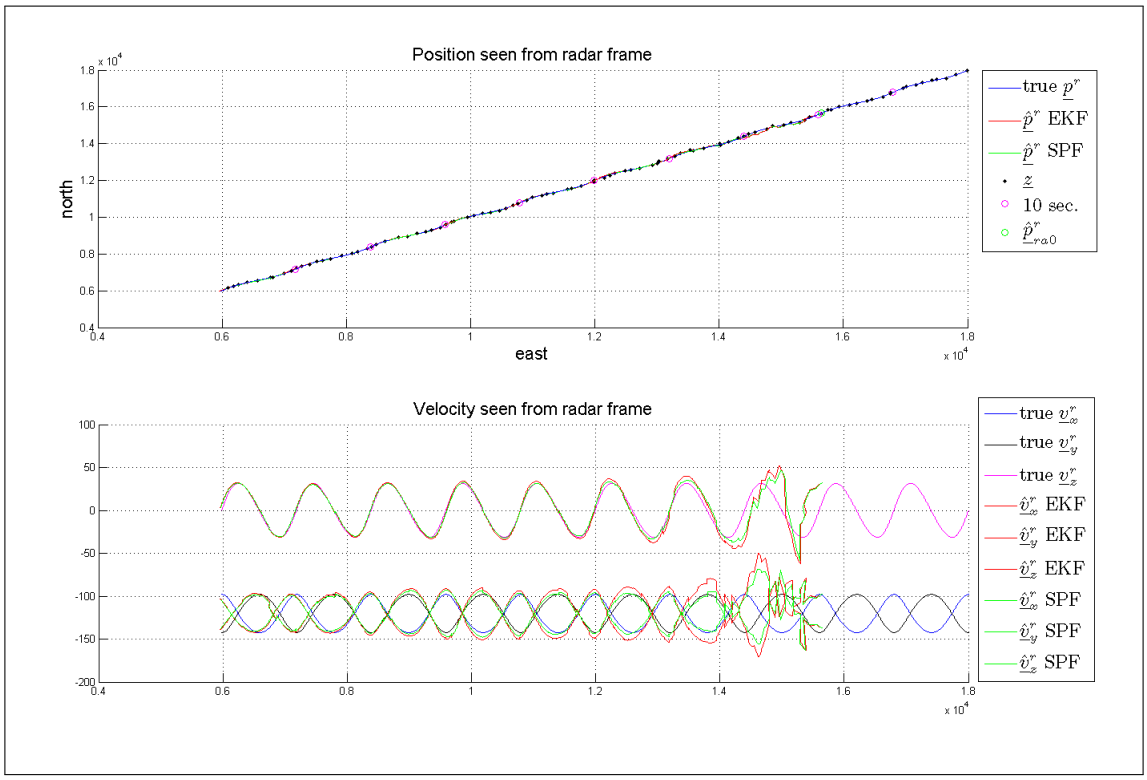

Figure 7.29: Scenario 2. Airplane position and velocity seen from $\mathcal{F}_r$. Perturbations known by the filters.

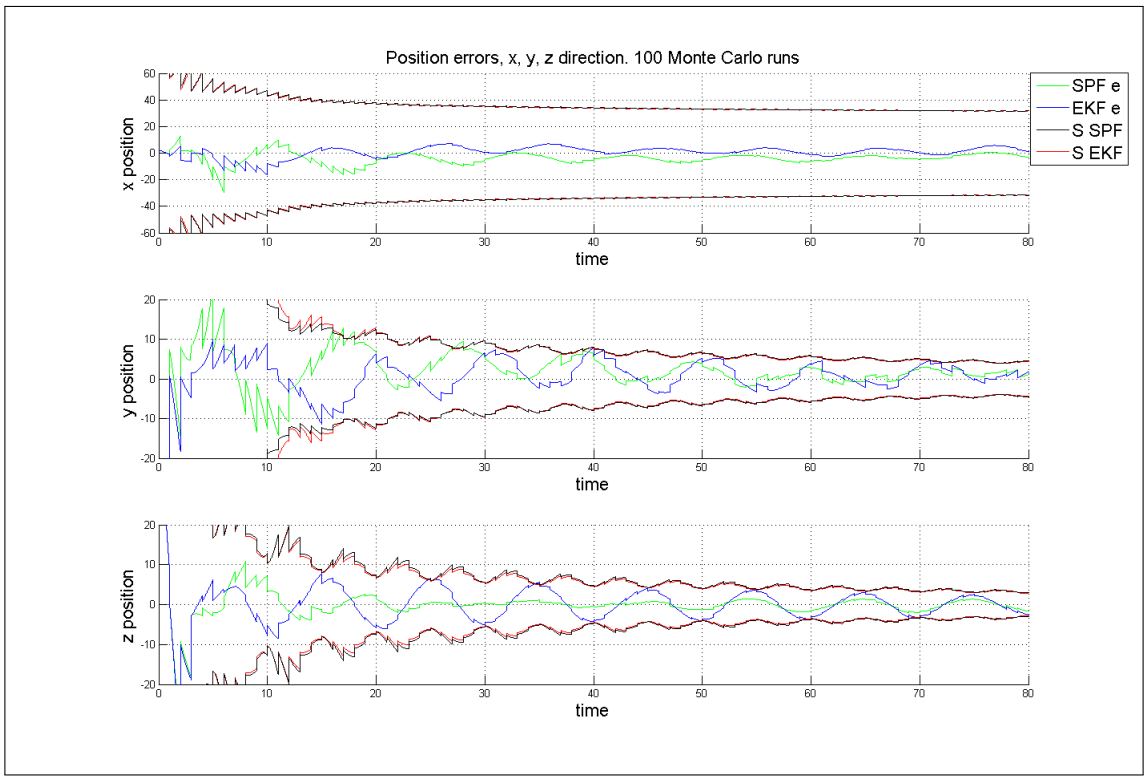

Figure 7.30: Scenario 2. Airplane position errors seen from $\mathcal{F}_r$. Perturbations known by the filters.

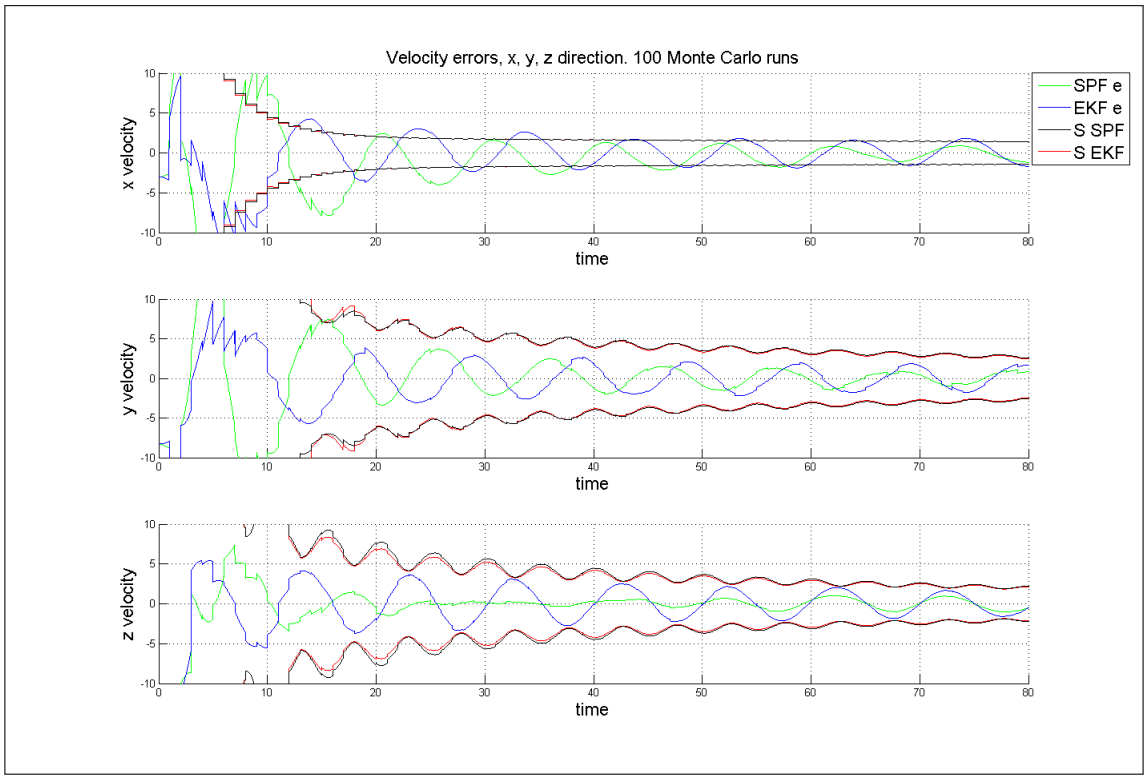

Figure 7.31: Scenario 2. Airplane velocity errors seen from $\mathcal{F}_r$. Perturbations known by the filters.

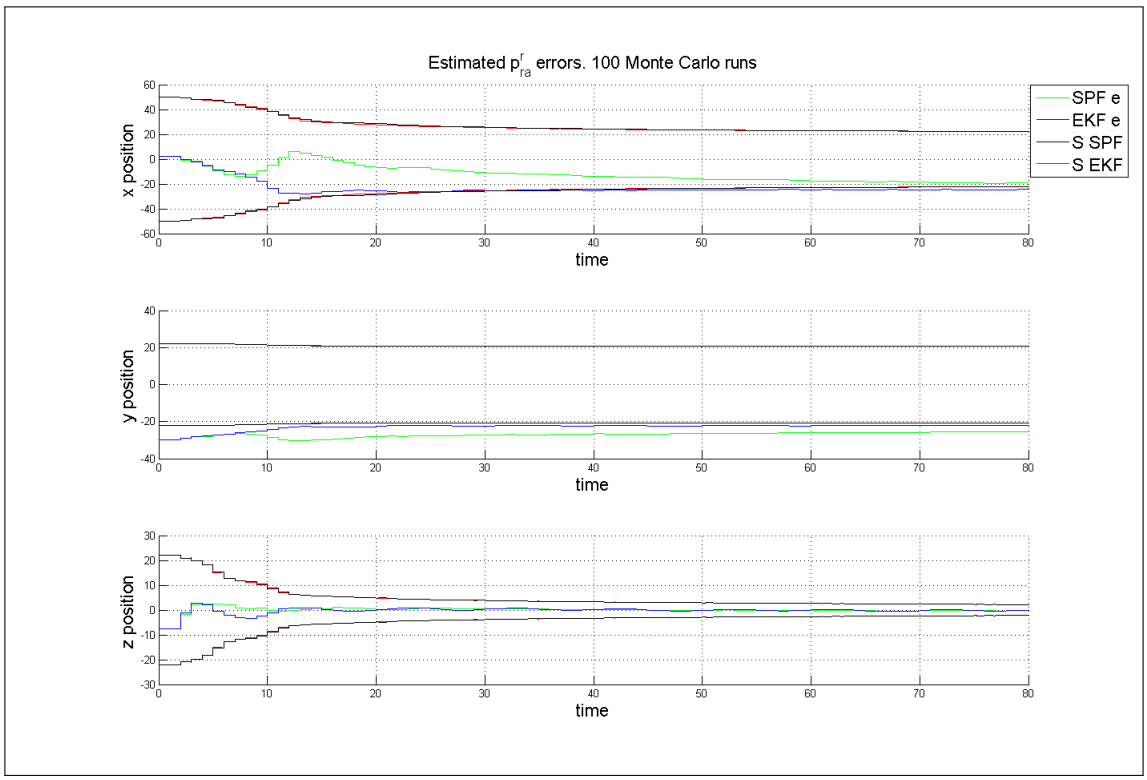

Figure 7.32: Scenario 2. Estimated $\underline{p}^r_{ra}$ errors seen from $\mathcal{F}_r$. Perturbations known by the filters.

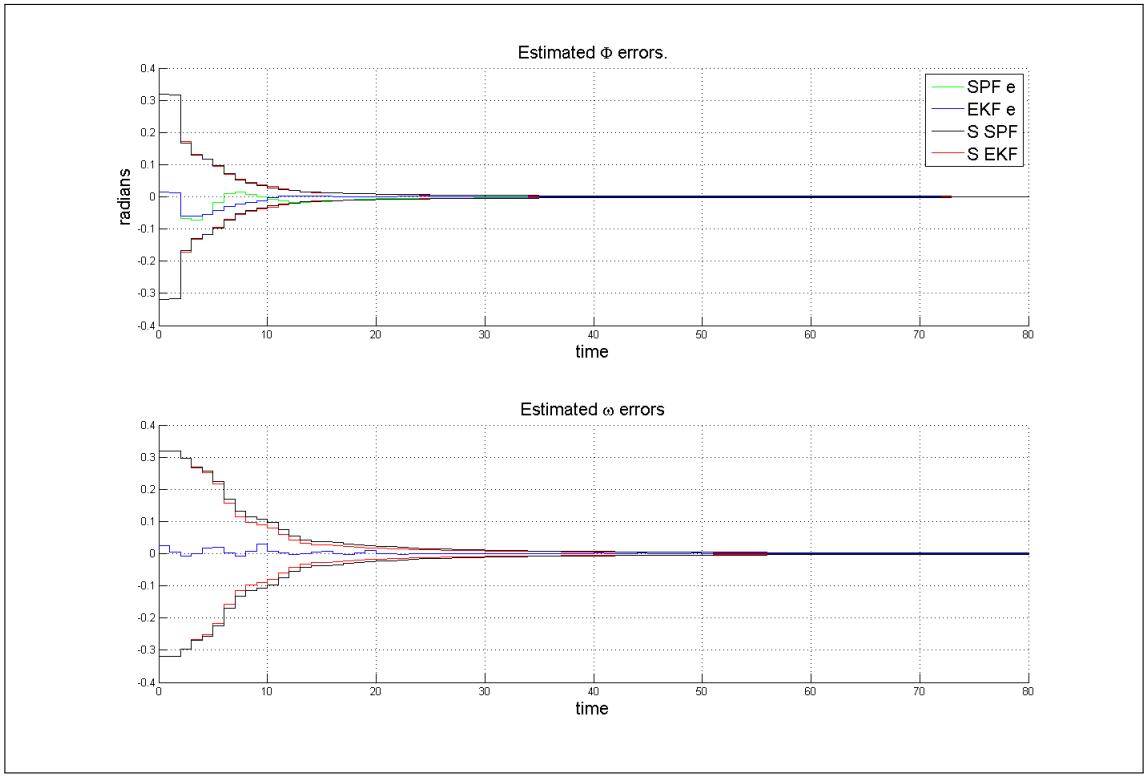

Figure 7.33: Scenario 2. Estimated ω and Φ errors. Perturbations known by the filters.

Scenario 3

In scenario 3 the airplane are heading away from the radar from north against east. Figure 7.34 shows position and velocity estimates for one single run. Figures 7.35 and 7.36 shows position and velocity errors for both filters. For position errors, the mean errors are below 5 meters for both filters. The SPF performes better than EKF in this scenario too. The same conclusions can be made looking at velocity mean errors in 7.36. There is no improvements in $\hat{\underline{p}}^r_{ra}$ in this scenario either. This is shown in figure 7.37, SPF is slightly better than EKF in x position, and EKF is sleightly better than SPF in y position. But both are giving unsatisfactory results. In z position SPF is faster than EKF converging to the true value. $\hat{\Phi}$ and $\hat{\omega}$ mean errors are shown in figure 7.38 are showing the same performance as in every other scenarios.

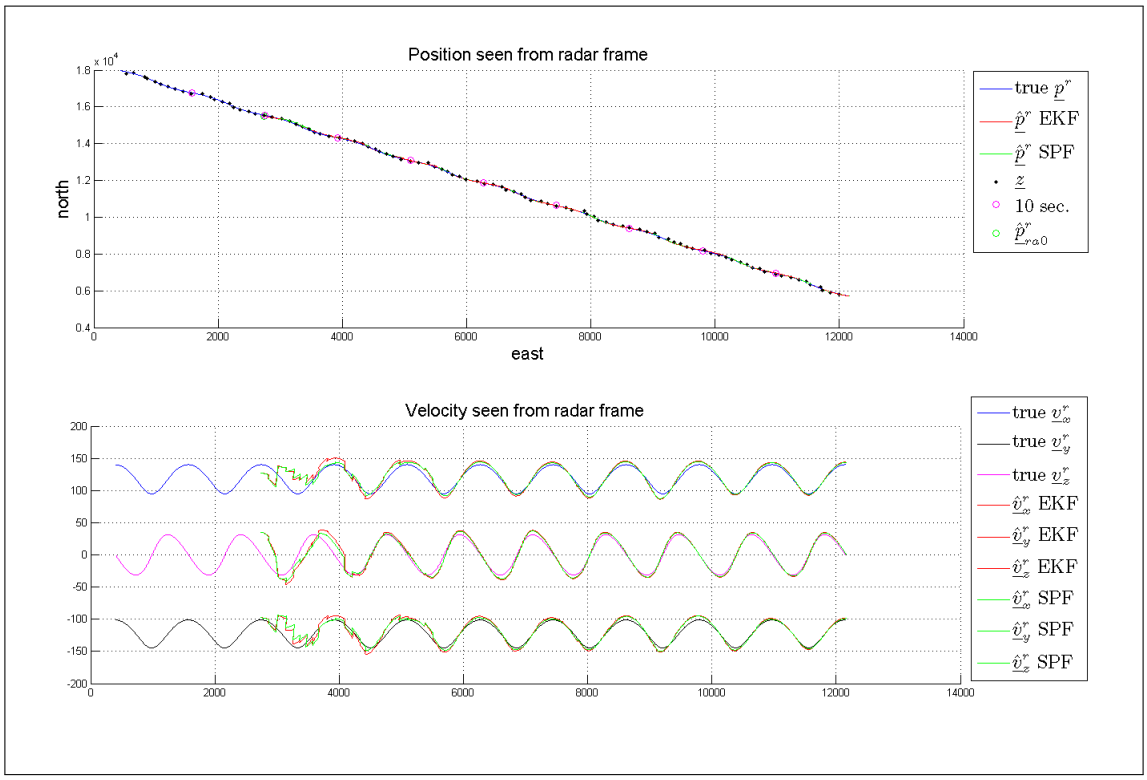

Figure 7.34: Scenario 3. Airplane position and velocity seen from $\mathcal{F}_r$. Perturbations known by the filters.

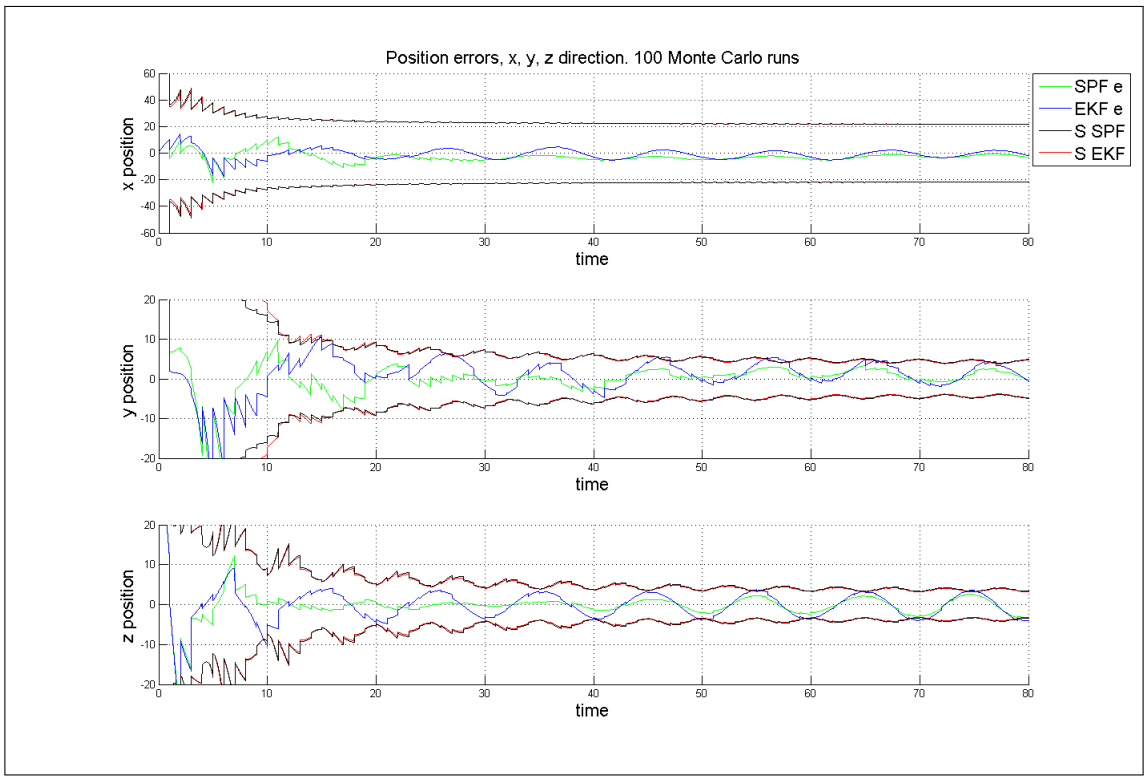

Figure 7.35: Scenario 3. Airplane position errors seen from $\mathcal{F}_r$. Perturbations known by the filters.

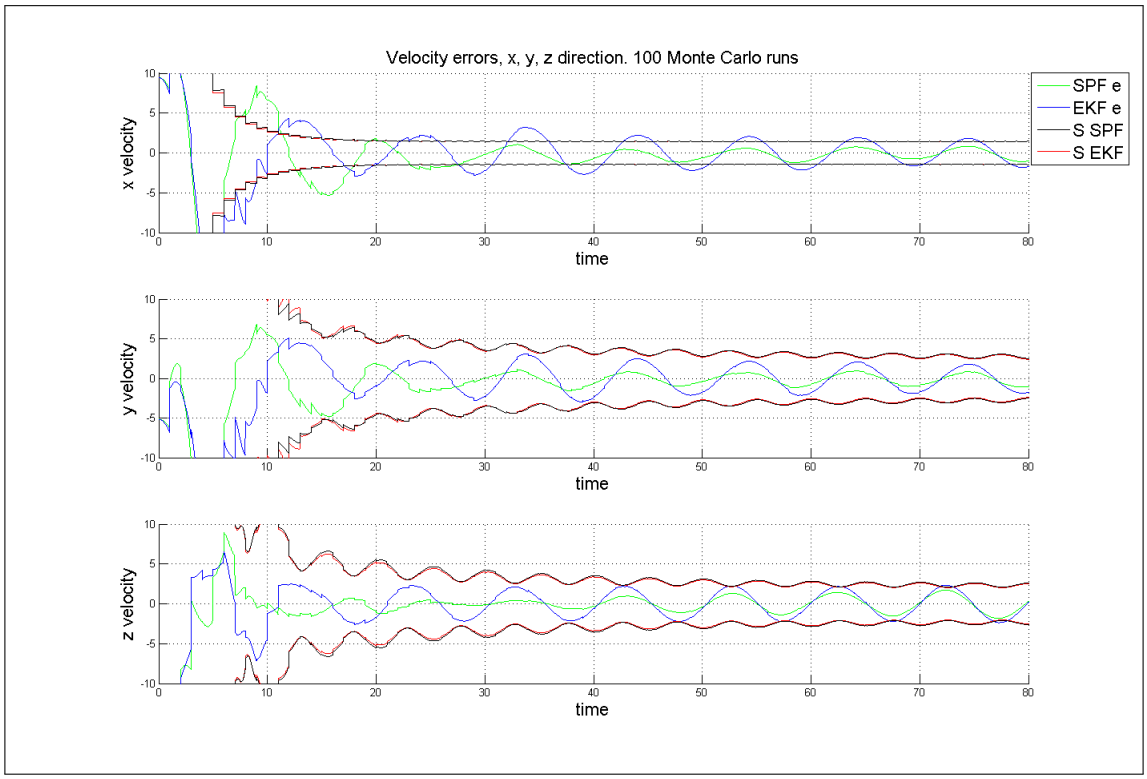

Figure 7.36: Scenario 3. Airplane velocity errors seen from $\mathcal{F}_r$. Perturbations known by the filters.

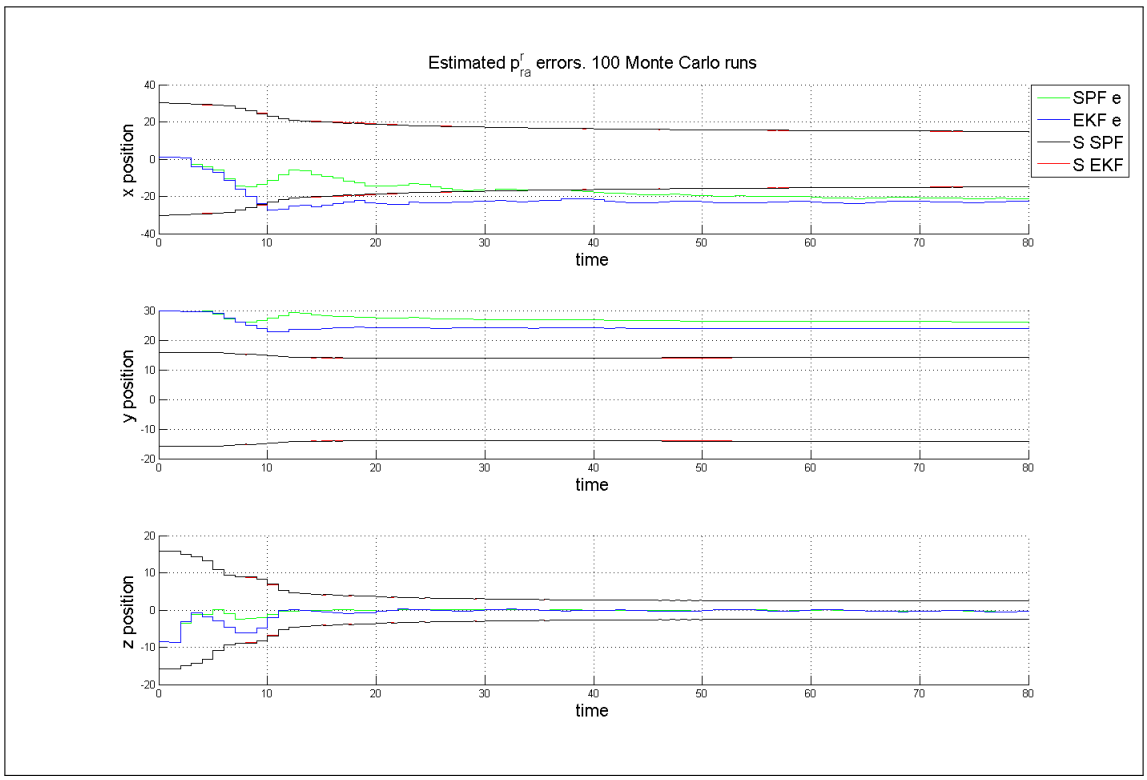

Figure 7.37: Scenario 3. Estimated $\underline{p}^r_{ra}$ errors seen from $\mathcal{F}_r$. Perturbations known by the filters.

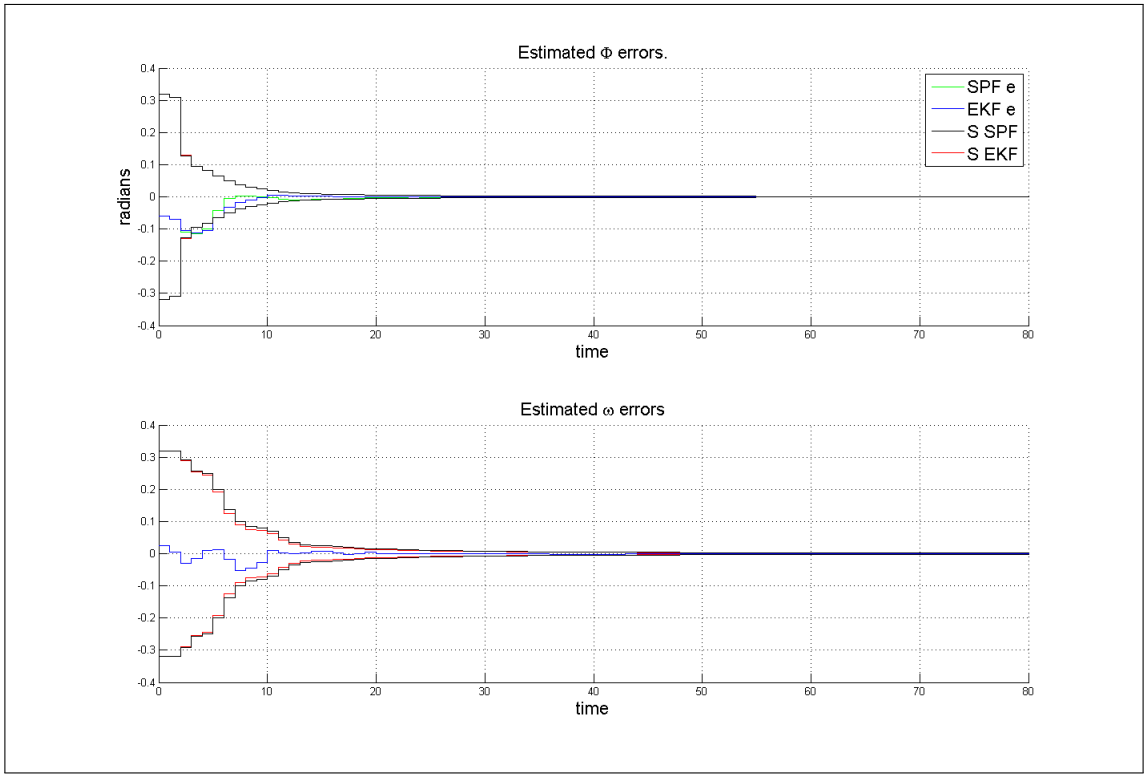

Figure 7.38: Scenario 2. Estimated ω and Φ errors. Perturbations known by the filters.

7.3 Falling body

In figure 7.39 position and velocity for one single run are shown. The blue line is the true position of the falling body, red and green line are EKF and SPF respectively. In the first seconds the velocity is almost constant, but as the falling body enters the atmosphere the velocity is decreasing with dense atmosphere. Due to this thicker air density at lower altitude, the gravity force are cancelled by a drag force, letting the falling body decrease its velocity. Looking at position in this figure it is not possible to see if one filter performed better than the other. Both filter plots lays completely on top of each other wich indicates equal performances. In figure 7.40 the system has been run with 100 Monte Carlo runs. Position errors, velocity errors and Ballistic constant errors are shown respectively. In the case of position errors, it is not large differences in performance. It may seem that SPF is slightly better from 20 seconds and out but thats negligible. In the case of velocity errors, EKF is slightly better from 0 to 15 seconds. But from here SPF performes better. But for the ballistic constant, SPF estimates the true value faster than EKF.

Similar falling body scenarios has been presented earlier in several sources, for example in [1, 4, 2]. In [1] there is no comparison of EKF and SPF, but it can be read that similar results are found by EKF. Especially the estimetion of the Ballistic constant, Arthur Gelb [1] states that EKF tracks it poor early in the trajectory because of the thin air at high altitude creates a small drag force on the body, and that the measurements includes little information of the ballistic constant. This emphasise the results found in this thesis.

In [4] the EKF and SPF are compared at almost exact the same system. Here it is shown that both filters have pikes of errors in velocity around 10 - 15 seconds, which also

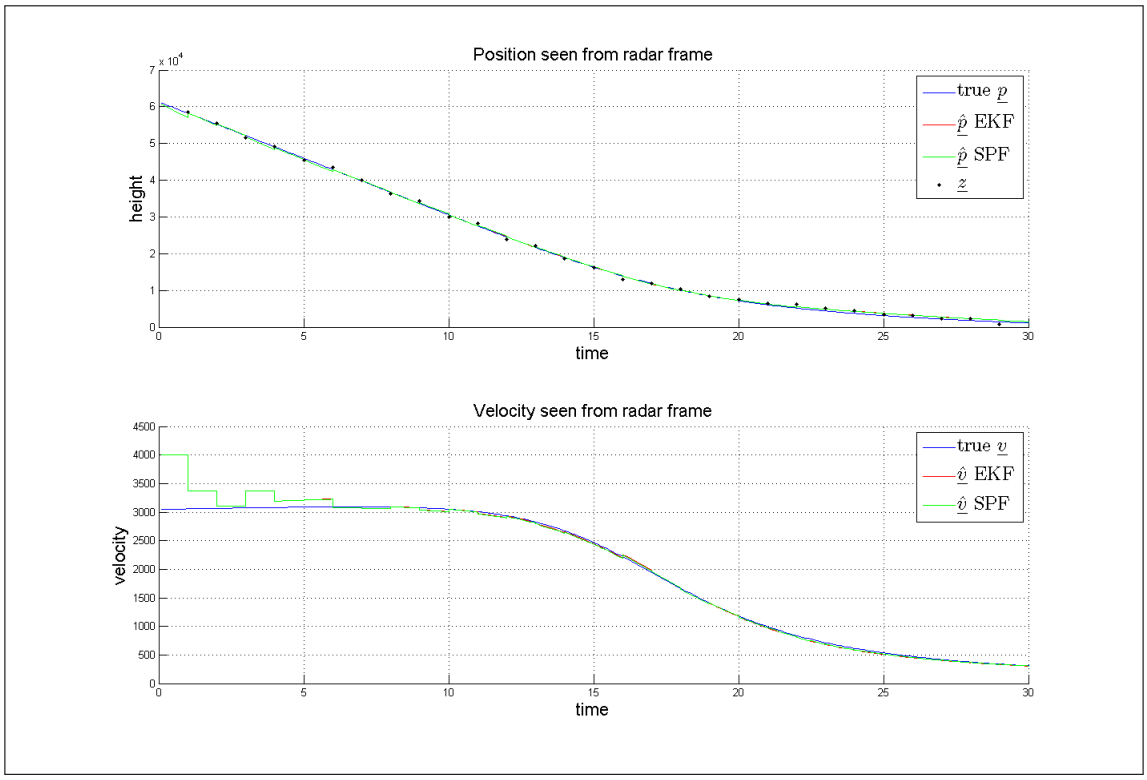

Figure 7.39: Falling body position and velocity seen from radar.

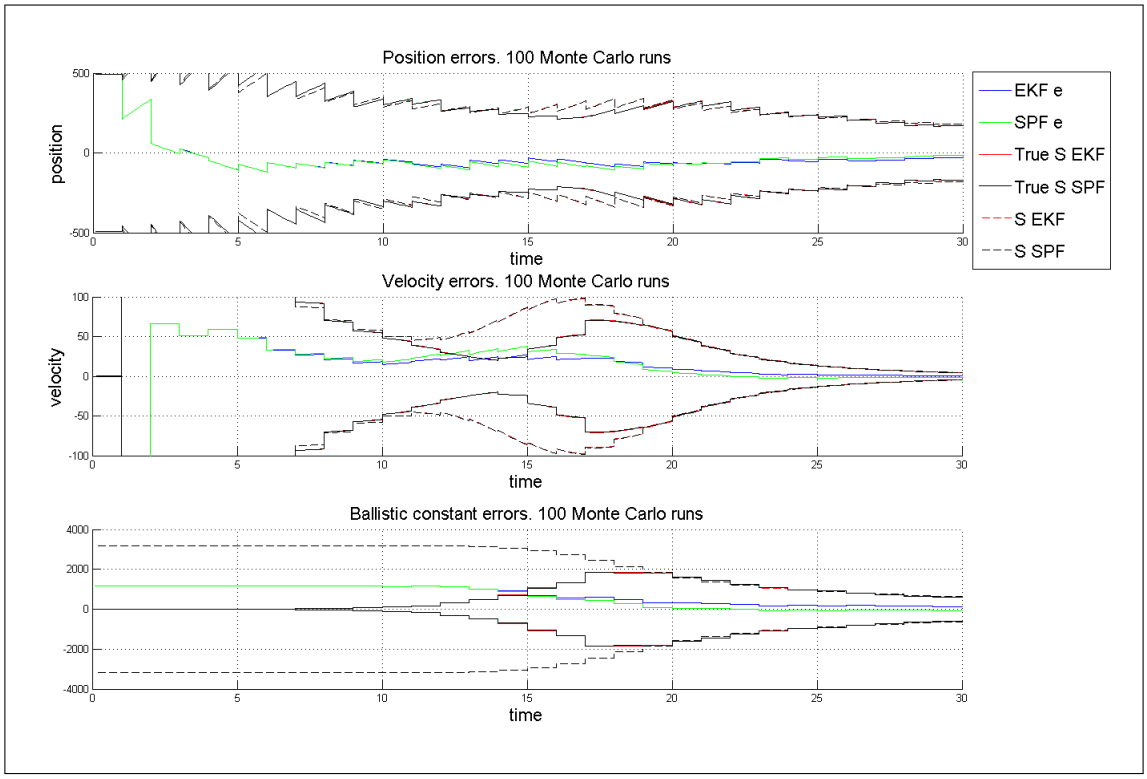

Figure 7.40: Falling body position errors, velocity errors and ballistic constant errors.

is the fact in figure 7.40, and that SPF consistently gives better estimates than EKF. In [2] the same conclusions as above are made.

Chapter 8

Conclusion

8.1 Discussion

This thesis shows a comparison of two filters, namely Extended Kalman filter (EKF) and Sigma Point filter (SPF). It is performed partly theoretical and by simulations. The scenarios that have been simulated are a falling body and two different airplane motions.

The airplane motions are divided in two cases. One having perturbations in the horizontal plane as a sine wave. The other has perturbations in the horizontal and vertical plane as a sine and cosine wave respectively. Each case are simulated and estimated by use of EKF and SPF receiving measurements from a radar in spherical coordinates. The filters are compared both by unknown and known airplane motions. In the case where the motions are unknown, a linear process filter model is implemented. This gave unsatisfactory results with high peaks of prediction. This would have been better using a more rapid measurement update. Since it is out of the scope implementing a more realistic radar model, a measurement update on 1 Hz where chosen by the author.

The tuning of the filters are a time-consuming job, some tuning where done on the process covariance matrix Q and $\hat{P}_0$. Here it could have been done a more structured analysis of both. SPF have even more tuning factors that could have been looked more into. Some tuning of these factors where done, but the author ended up using them as described in the theory of this thesis. Partly because it gave good results, and in theory should be the best choice used on stochastic variables of additive white noise.

In the case where the airplane motions are known by the filters, a non-linear process filter model is implemented. A goal for the author was to implement a model that worked well on incomming airplanes from all directions within north and east. Several models where implemented and tried. It ended up with having a more complex measurement matrix in the filters. This was an error prone process transforming to spherical coordinates in the measurement equation and then performe linearization for use in EKF. For the non-linear process model case, the measurements are therefore converted to cartesian coordinates before they are implemented in the filters. As the author of this thesis have

experienced by reading, this is not the usual way to do it. The transformations are generally done in the measurement matrix.

The MUSIC estimate is used directly in the measurement update exchanging the Kalman filter estimate with the MUSIC estimate. This estimate could have been received as a measurement in the filters, which may have given better covariance update, since there may be crosscorrelations that could have improved other estimates.

The falling body is modelled as an object falling towards the ground from high altitude, straight against a radar on the ground. The radar gives measurements in cartesian coordinates, giving a non-linear process filter model and a linear measurement equation. This where implemented without use of process covariance matrix Q in the filters. The implementation of Q in the filters may have improved the filters performance as it usually do.

8.2 Conclusion

In this thesis Extended Kalman filter (EKF) have been compared with the more recent Sigma Point filter (SPF). The comparisons have been done on target tracking of an airplane moving in horizontal wave motion and helix motion where the filters receives measurements in spherical coordinates. This have been done where the process is unknown and known for the filters. Three scenarios where looked into, where the airplane is heading straight against the radar from east, straight against the radar from north east, and away from the radar from north against east. In addition tracking of an object falling towards the ground from high altitude have been carried out. Theory of the filters have been presented and implemented by simulations in Matlab.

According the theory shall the Sigma Point filter performe better than the Extended Kalman filter. The results in this thesis estimating the airplane state vectors where the process is known for the filters, SPF showed all over better results than EKF estimating position and velocity. Estimation of $\underline{p}^r_{ra}$, the vector defining origin in the airplane frame represented in radar frame, showed various results for both filters. Heading angle $\hat{\Phi}$ was estimated with equally results for both filters. The same conclusion can be made of the angular velocity $\hat{\omega}$. This where estimated using MUSIC since both filters had unsatisfactory properties estimating the frequency needed to calculate the angular velocity. So equal results estimating ω where assumed. This was shown for all of the three scenarios for non-linear process filter model.

In the case where the process is unknown for the filters only position and velocity where estimated and a linear process filter modell where used. An important observation is that SPF showed better results in most of the scenarios. The same conclusion can be made in the case of the falling body. The conclusions are based on mean error calculations.

8.3 Recommendations

The results given in this thesis are for target tracking between north and east. A study on performance in all four quadrants with different measurement noise could be a interesting task. Increased measurement noise can have a brutal effect on the MUSIC estimate and then decrease the performance of the non-linear filter models drastical. Some future work could be finding a way to estimate frequency that are less sensitive to measurement noise, and even can handle other types of noise, than additive white noise.

Implementing of more realistic types of measurement noise is an important aspect of target tracking using radars. In this thesis additive white noise have been used. Several other models of radar noise can be simulated according to a model of a real radar.

Horizontal wave motion and helix motion have been handled in this thesis. These two can be used together meaning that it can be switched between the two trajectories by random. Then a maneuver detection algorithm is needed to switch between process models in the filters. This can also be extended to dealing with circular turn maneuvers for instance turns of 90 or 180 degrees.

To have a more accurate estimate of the tracking process, sensor fusion can be applied. This can be done using two or more sensors and compare the estimation and prediction properties of the filters when they recieve redundant measurements.

Appendix A

Derivation of the Kalman Filter Equations

This derivation of the Kalman filter equations is taken from [1]. The goal is to estimate the states $\underline{x}_k$ based on the measurements $\underline{z}_k$ and the knowledge of the system dynamics 3.1. To predict the estimate of $\underline{x}_k$ based on measurements up to time t_{k+1} at current time t_k we have the a priori estimate

$$\begin{aligned} E[\underline{x}_{k+1}] &= E[\Phi\underline{x}_k + \Lambda\underline{u}_k + \Gamma\underline{v}_k] \\ &\Downarrow \\ \bar{\underline{x}}_{k+1} &= \Phi\bar{\underline{x}}_k + \Lambda\underline{u}_k \end{aligned} \tag{A.1}$$

This is the time update for $\underline{x}$ which updates only based on our knowledge of the system after the last measurements. The error covariance matrix is defined as

$$P_k = E[\bar{\underline{e}} - \bar{\underline{e}}^T] = E[(\underline{x}_k - \bar{\underline{x}}_k)(\underline{x}_k - \bar{\underline{x}}_k)^T] \tag{A.2}$$

Then the calculation of the prdicted error covariance, $\bar{P}_{k+1}$, can be done by subsituting equations 3.1 and A.1 into A.2

$$\begin{aligned} \bar{P}_{k+1} &= E[(\Phi\bar{\underline{x}}_k + \Lambda\underline{u}_k + \Gamma\underline{v}_k - \Phi\bar{\underline{x}}_k - \Lambda\underline{u}_k)(\cdot)^T] \\ &= E[(\Phi(\underline{x}_k\bar{\underline{x}}_k) + \Gamma\underline{v}_k)(\cdot)^T] \\ &= \Phi_k\bar{P}_k\Phi_k^T + \Gamma_k Q_k\Gamma_k^T \end{aligned} \tag{A.3}$$

In the following paragraph we will compute the measurement update equations for the Kalman filter, which is taken from [1]. Consider the linear stochastic discrete time system equation

$$\underline{x}_{k+1} = \Phi_k\underline{x}_k + \underline{v}_k \tag{A.4}$$

and with measurement equation

$$\underline{z}_k = H_k\underline{x}_k + \underline{v}_k \tag{A.5}$$

with the same properties as in 3.3. We have already computed the a priori estimate $\bar{\underline{x}}_{k+1}$. Given this, we seek an updated estimate $\hat{\underline{x}}_k$ based on all the measurements in $\underline{z}_k$. In

recursive form we have

$$\underline{\hat{x}}_k = K'_k \underline{\bar{x}}_k + K_k \underline{z}_k \tag{A.6}$$

where K'_k and K_k are time variant weighting matrices, that still are unspecified. An equation for estimation error can be computed from equation A.6 inserted in equation A.5, and by defining

$$\begin{aligned} \underline{\hat{x}}_k &= \underline{x}_k + \underline{\hat{e}}_k \\ \underline{\bar{x}}_k &= \underline{x}_k + \underline{\bar{e}}_k \end{aligned} \tag{A.7}$$

we get

$$\begin{aligned} \underline{\hat{x}}_k &= K'_k \underline{\bar{x}}_k + K_k H_k \underline{x}_k + K_k \underline{v}_k \\ \underline{\hat{e}}_k &= K'_k \underline{\bar{x}}_k + K_k H_k \underline{x}_k + K_k \underline{v}_k - \underline{x}_k \\ \underline{\hat{e}}_k &= (K'_k + K_k H_k - I)\underline{x}_k + K'_k \underline{\bar{e}}_k + K_k \underline{v}_k \end{aligned} \tag{A.8}$$

As we already have seen we have by definition that $E[\underline{v}_k] = \underline{0}$. If this estimator shall be unbiased (that is $E[\underline{\hat{e}}_k] = \underline{0}$) for any given state vactor $\underline{x}_k$, it is required that $(K'_k + K_k H_k - I)$ in A.8 equals zero. Thus,

$$K'_k = I - K_k H_k \tag{A.9}$$

and we can compute the estimate of $\underline{x}_k$

$$\begin{aligned} \underline{\hat{e}}_k &= (K'_k + K_k H_k - I)\underline{x}_k + K'_k \underline{\bar{e}}_k + K_k \underline{v}_k \\ \underline{0} &= [I - K_k H_k + K_k H_k - I]\underline{x}_k + \underline{\bar{e}}_k - K_k H_k \underline{\bar{e}}_k + K_k \underline{v}_k \\ \underline{0} &= \underline{\bar{e}}_k - K_k H_k \underline{\bar{e}}_k + K_k \underline{v}_k \\ \underline{0} &= -\underline{x}_k + (I - K_k H_k)\underline{\bar{x}}_k + K_k \underline{z}_k \\ &\Downarrow \\ \underline{\hat{x}}_k &= (I - K_k H_k)\underline{\bar{x}}_k + K_k \underline{z}_k \end{aligned} \tag{A.10}$$

or

$$\underline{\hat{x}}_k = \underline{\bar{x}}_k + K_k(\underline{z}_k - H_k \underline{\bar{x}}_k) \tag{A.11}$$

This is the estimate of $\underline{x}_k$. If we use the corrosponding estimation error equation A.8, we can compute the equation for change in error covariance matrix when a measurement is employed. From definition we have that

$$\hat{P} = E[\underline{\hat{e}}_k \underline{\hat{e}}_k^T] \tag{A.12}$$

If we substitude this into equation A.8 for the estimation error, we get

$$\begin{aligned} \hat{P}_k &= E[\Big((I - K_k H_k)\underline{\bar{e}}_k + K_k \underline{v}_k\Big)\Big((I - K_k H_k)\underline{\bar{e}}_k + K_k \underline{v}_k\Big)^T] \\ \hat{P}_k &= E[(I - K_k H_k)\underline{\bar{e}}_k + K_k \underline{v}_k\Big(\underline{\bar{e}}_k^T (I - K_k H_k)^T + \underline{v}_k^T K_k^T\Big) + K_k \underline{v}_k\Big(\underline{\bar{e}}_k^T (I - K_k H_k)^T + \underline{v}_k K_k^T\Big)] \end{aligned} \tag{A.13}$$

From the definitions we have

$$\begin{aligned} E[\underline{\bar{e}}_k \underline{\bar{e}}_k^T] &= \bar{P}_k \\ E[\underline{v}_k \underline{v}_k^T] &= R_k \end{aligned} \tag{A.14}$$

Using this in equation A.13 we get

$$\hat{P}_k = (I - K_k H_k)\bar{P}_k(I - K_k H_k)^T + K_k R_k K_k^T \tag{A.15}$$

To get the optimum choice of K_k we require to minimize a weighted scalar sum of the diagonal elements of the error covariance matrix $\hat{P}_k$. Thus, the choice of the cost function will be

$$J_k = E[\underline{\hat{e}}^T S \underline{\hat{e}}] \tag{A.16}$$

where S is any positive semidefinite matrix. To get the value of K_k that gives a minimum, it is necessary to find the partial derivative of J_k with respect to K_k, which then shall be equal to zero. Use of the mathematical rule

$$\frac{\delta}{\delta A}[trace(ABA^T)] = 2AB \tag{A.17}$$

where A and B are matrices and B is symmetric, we get from equations A.15 and A.16 solved for K_k

$$\begin{gathered} \tfrac{\delta}{\delta K_k}[trace\Big((I - K_k H_k)\bar{P}_k(I - K_k H_k)^T + K_k R_k K_k^T\Big)] = 0 \\ -2(I - K_k H_k)\bar{P}_k H_k^T + 2K_k R_k = 0 \\ K_k = \bar{P}_k H_k^T (H_k \bar{P} H_k^T + R_k)^{-1} \end{gathered} \tag{A.18}$$

This is called the Kalman gain. Substituting this into equation A.15 gives the optimized value of the updated estimation error covariance matrix.

$$\hat{P}_k(I - K_k H_k)\bar{P}_k \tag{A.19}$$

Now all of the descrete Kalman filter equations are computed. This can be summarized as followed

$$\underline{\bar{x}}_{k+1} = \Phi_k \underline{\hat{x}}_k + \Lambda_k \underline{u}_k \tag{A.20}$$

$$\bar{P}_{k+1} = \Phi_k \hat{P}_k \Phi_k + \Gamma_k Q_k \Gamma_k^T \tag{A.21}$$

$$\underline{\hat{x}}_k = \underline{\bar{x}}_k + K_k(\underline{z}_k - H_k \underline{\bar{x}}_k) \tag{A.22}$$

$$K_k = \bar{P}_k H_k^T (H_k \bar{P}_k H_k^T + R_k)^{-1} \tag{A.23}$$

$$\hat{P}_k = (I - K_k H_k)\bar{P}_k \tag{A.24}$$

Appendix B

Simulation Results for Linear Process Filter Models

B.1 Horizontal Wave Motion

Scenario 2

Figure B.1 shows estimated position and velocities for both EKF and SPF for one single run. The figure of plotted position do not say much of the quality of the estimation, it only shows the scenario where the airplane are heading straight towards the radar from north east. The figure of velocity is not easy to read either, but we can see hat SPF has higher peaks than EKF. For more details of performance, we must look at figure B.2 and B.3.

Figure B.2 and B.3 shows position and velocity errors for all three directions. Here we can see that SPF performs better than EKF in every directions for both position and velocities.

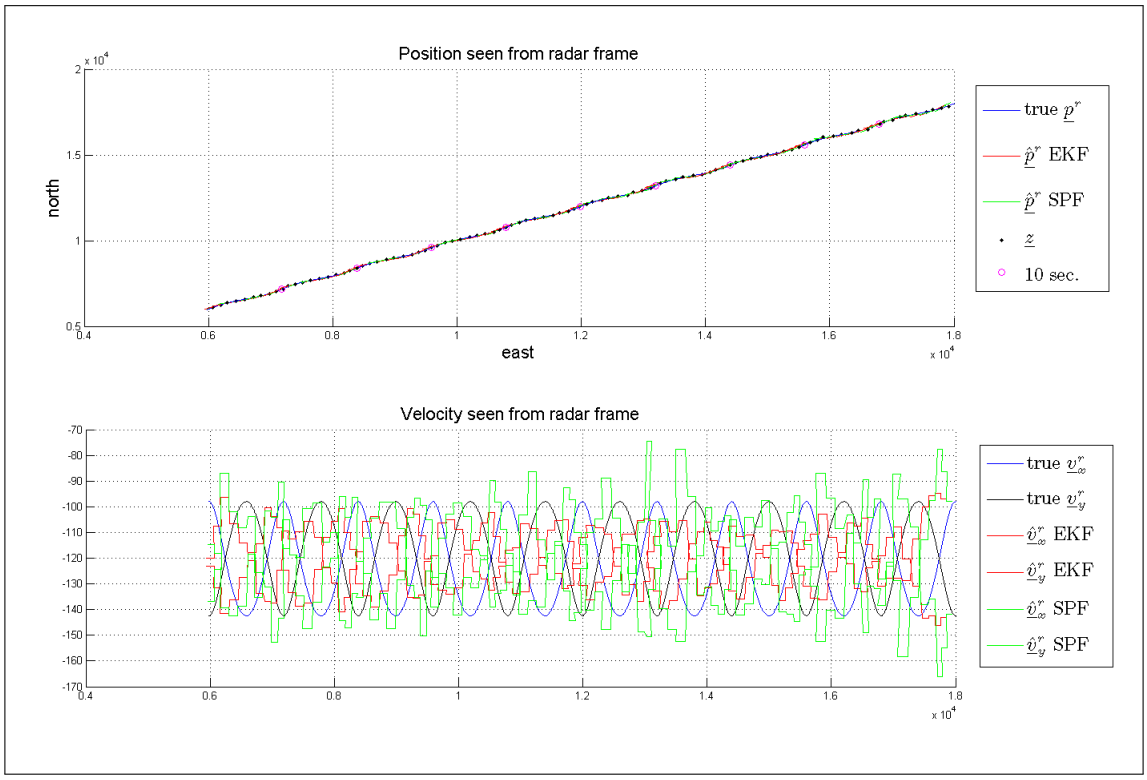

Figure B.1: Scenario 2, wave motion. Airplane position and velocities seen from $\mathcal{F}_r$. Perturbations unknown for the filters.

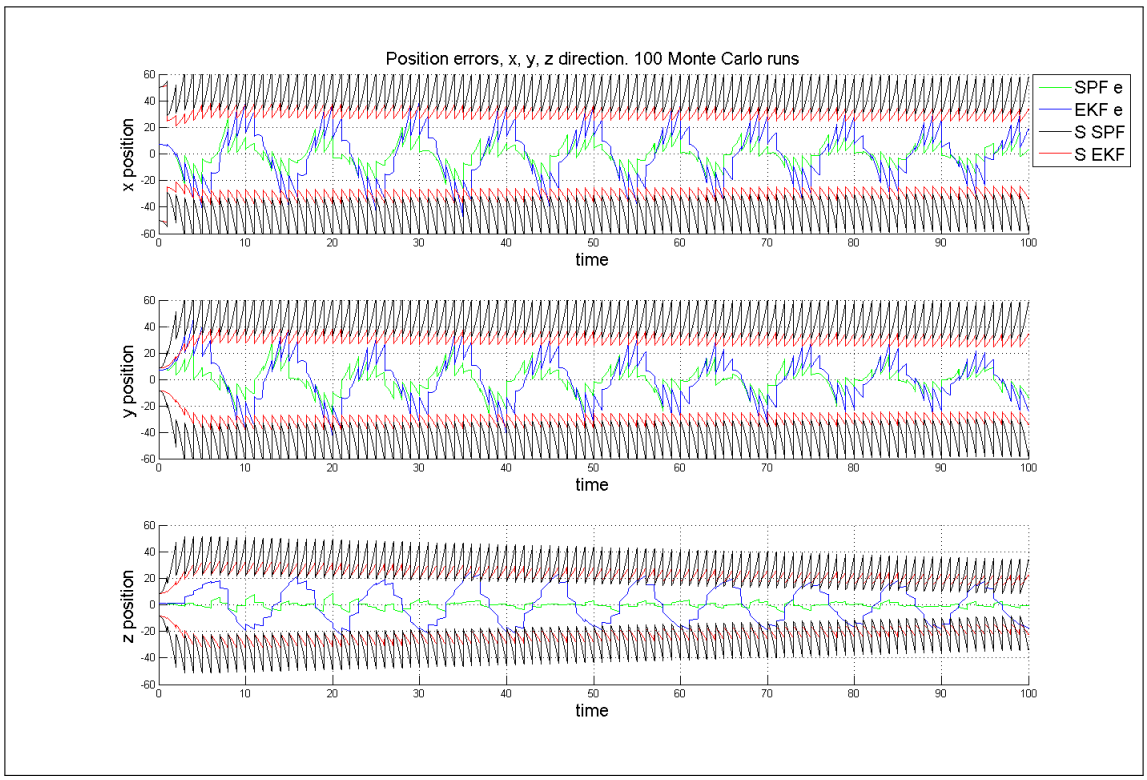

Figure B.2: Scenario 2, wave motion. Airplane position errors seen from $\mathcal{F}_r$. Perturbations unknown for the filters.

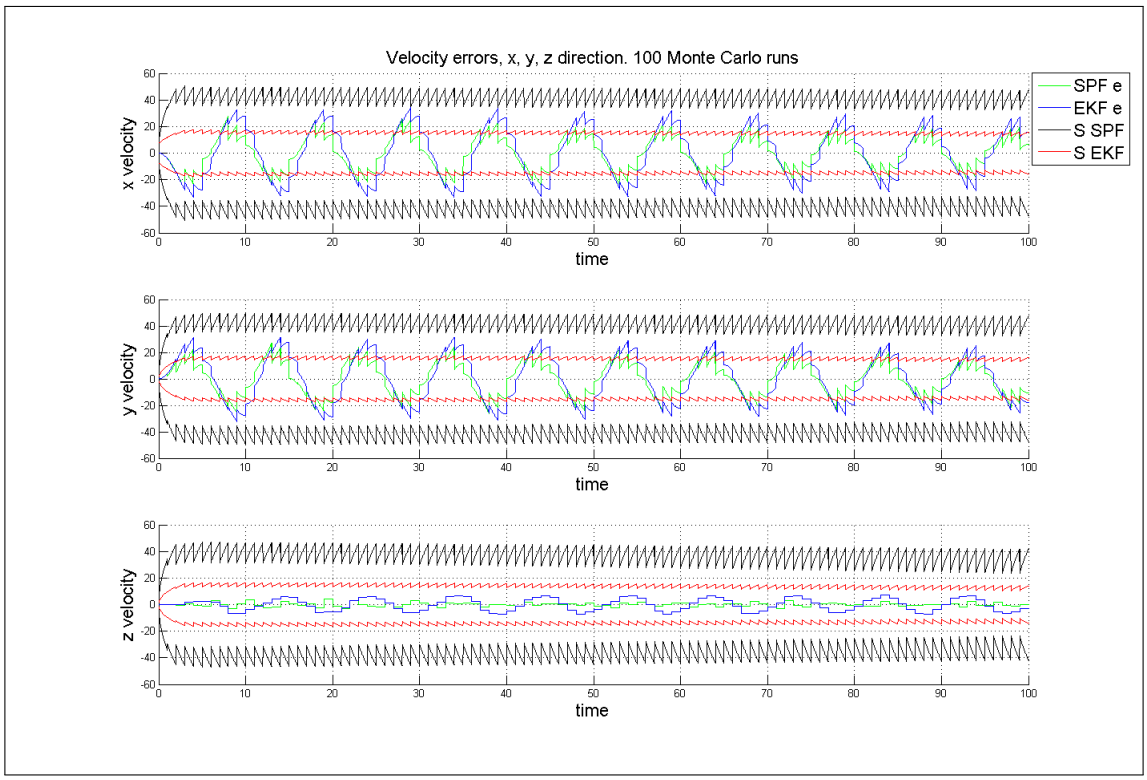

Figure B.3: Scenario 2, wave motion. Airplane velocity errors seen from $\mathcal{F}_r$. Perturbations unknown for the filters.

Scenario 3

B.2 Helix Motion

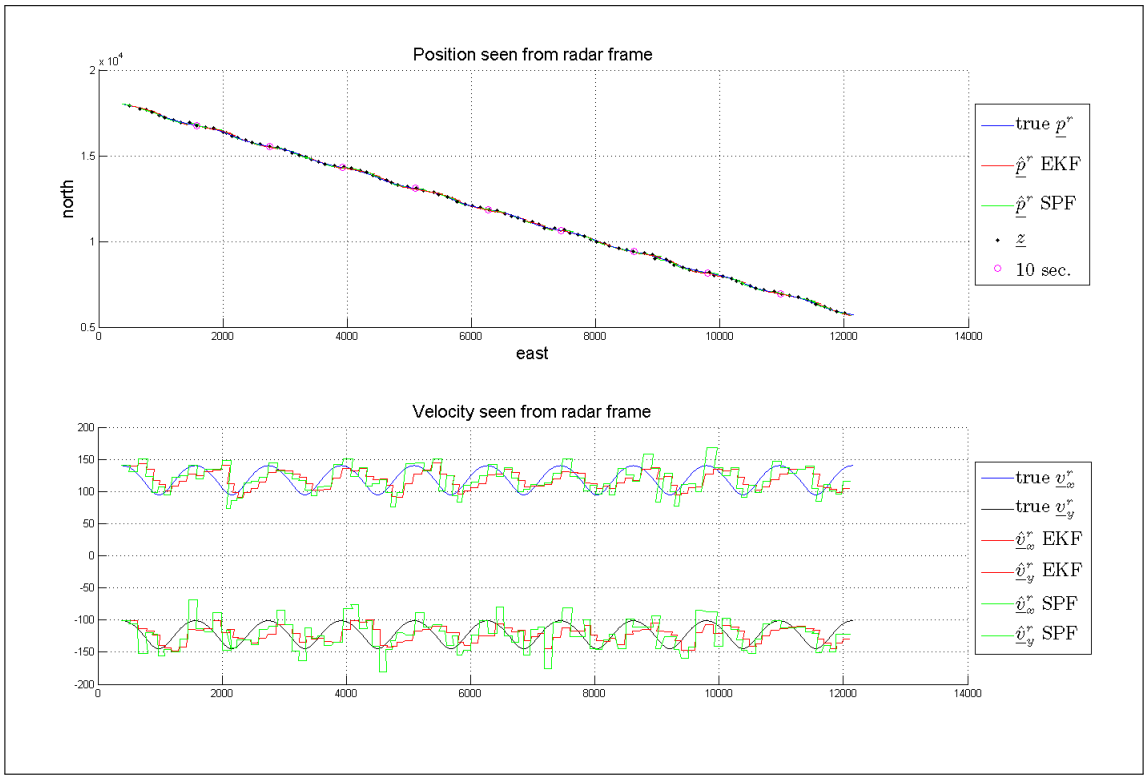

Figure B.4: Scenario 3, wave motion. Airplane position and velocities seen from $\mathcal{F}_r$. Perturbations unknown for the filters.

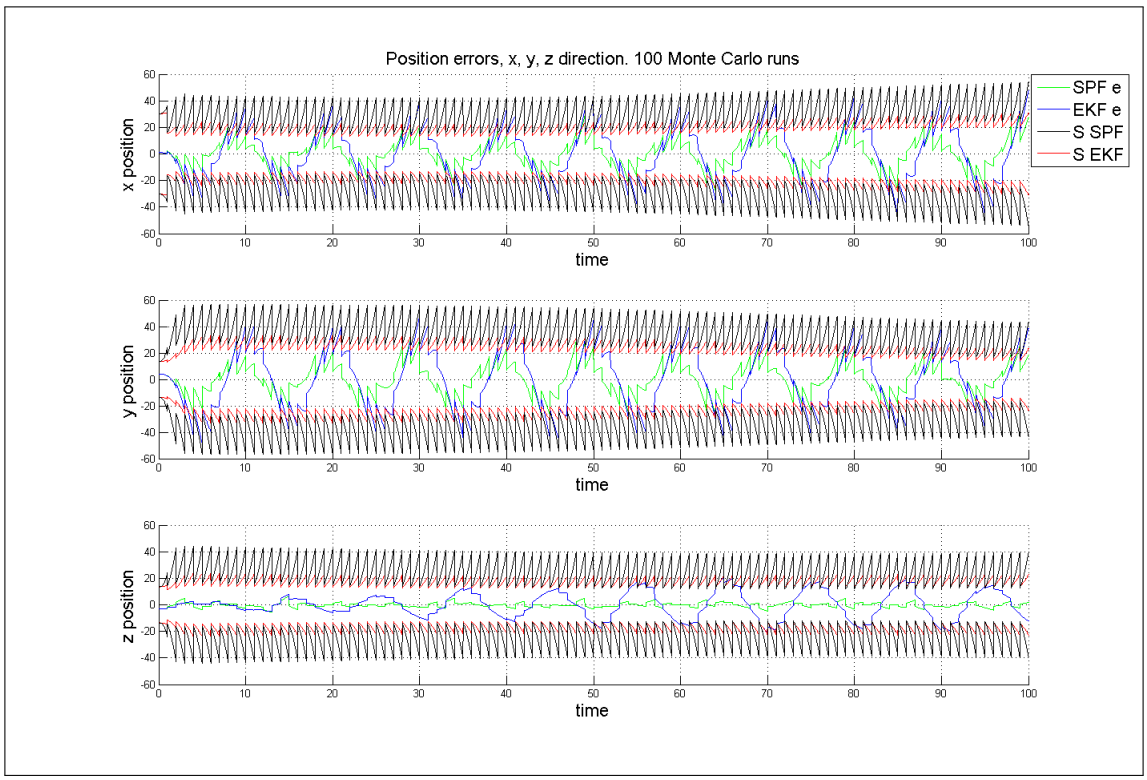

Figure B.5: Scenario 3, wave motion. Airplane position errors seen from $\mathcal{F}_r$. Perturbations unknown for the filters.

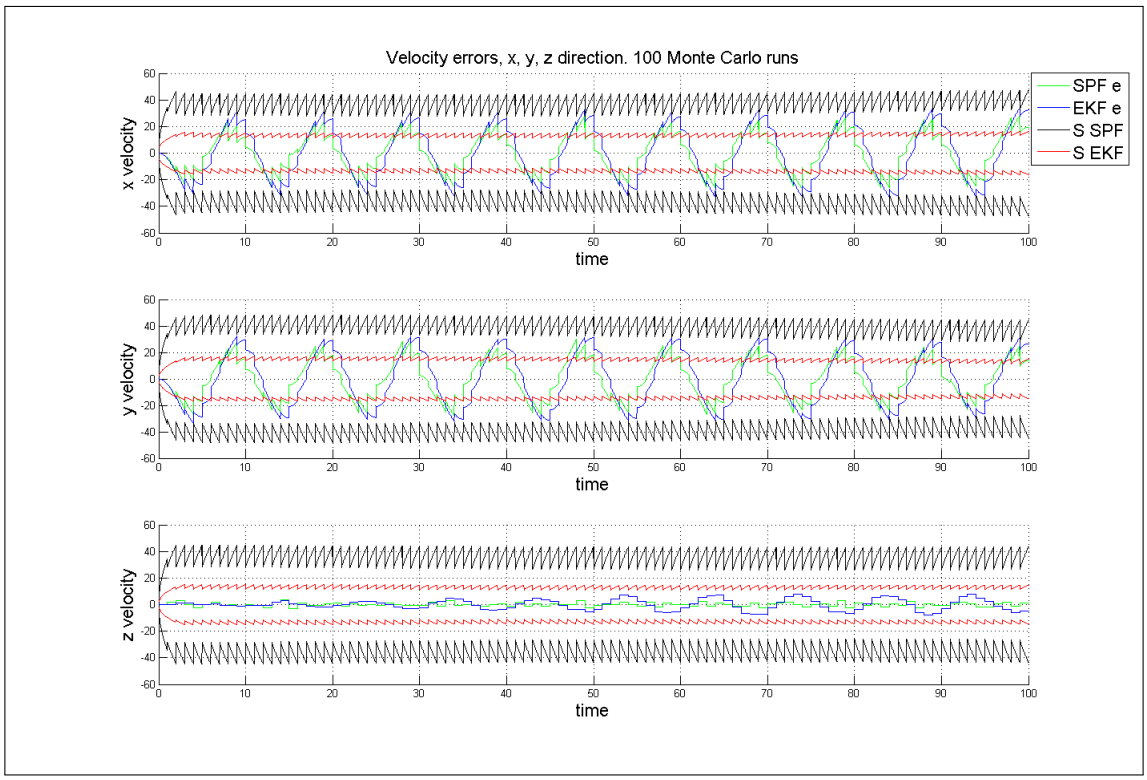

Figure B.6: Scenario 3, wave motion. Airplane velocity errors seen from $\mathcal{F}_r$. Perturbations unknown for the filters.

Scenario 2.

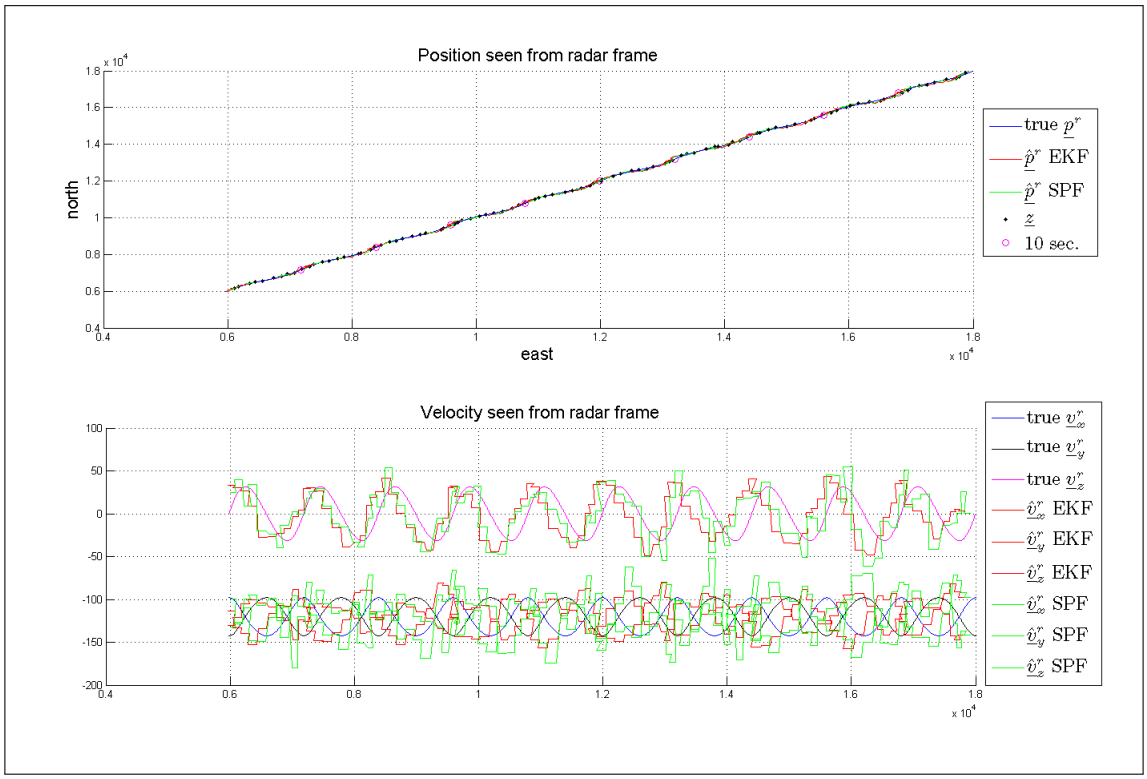

Figure B.7: Scenario 2, helix motion. Airplane position and velocities seen from $\mathcal{F}_r$. Perturbations unknown for the filters.

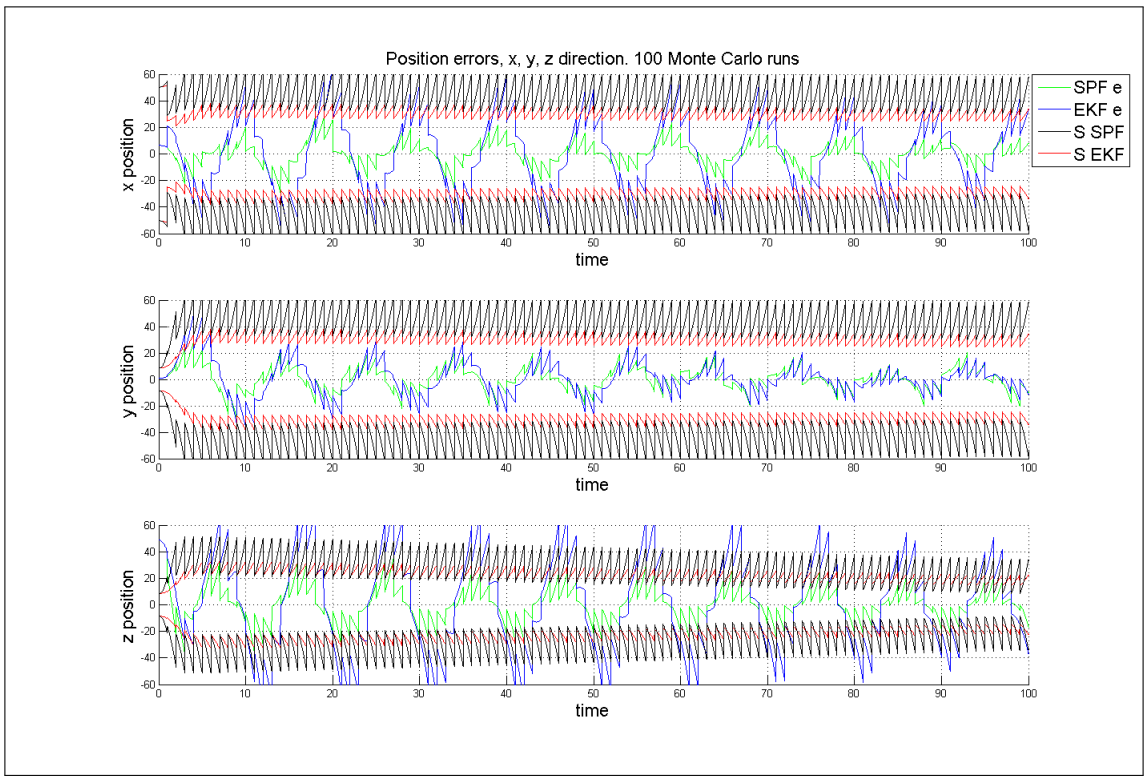

Figure B.8: Scenario 2, helix motion. Airplane position errors seen from $\mathcal{F}_r$. Perturbations unknown for the filters.

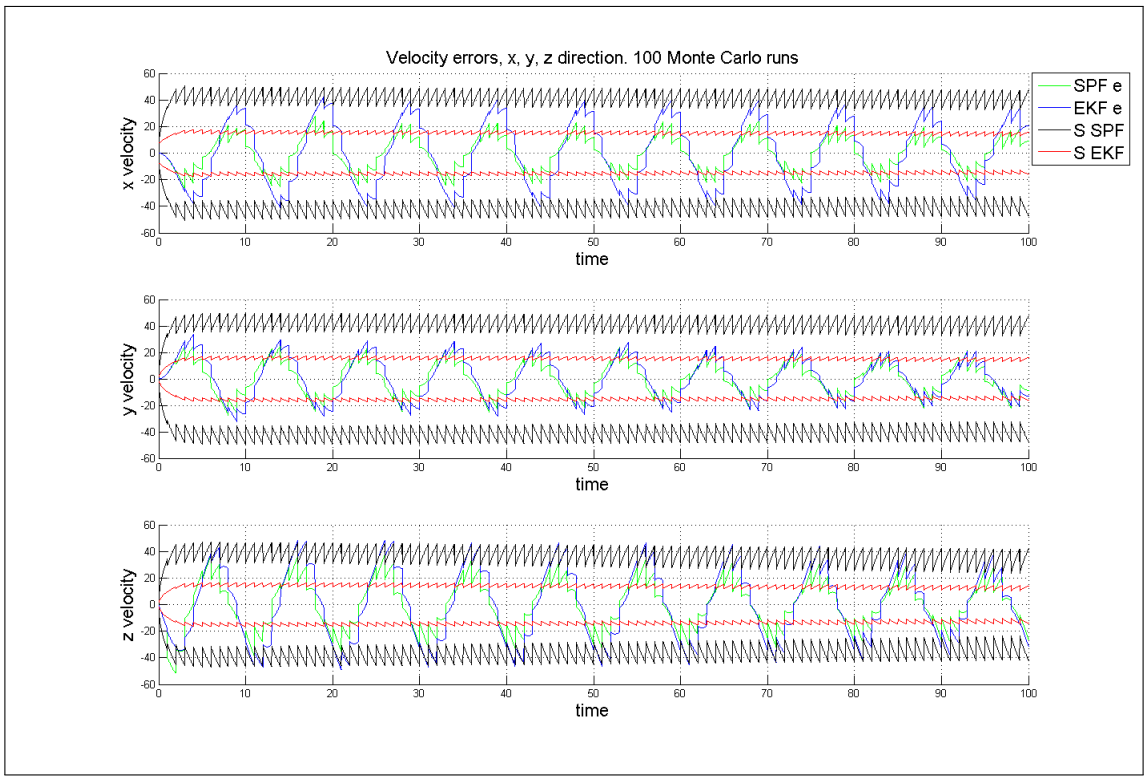

Figure B.9: Scenario 2, helix motion. Airplane velocity errors seen from $\mathcal{F}_r$. Perturbations unknown for the filters.

Scenario 3.

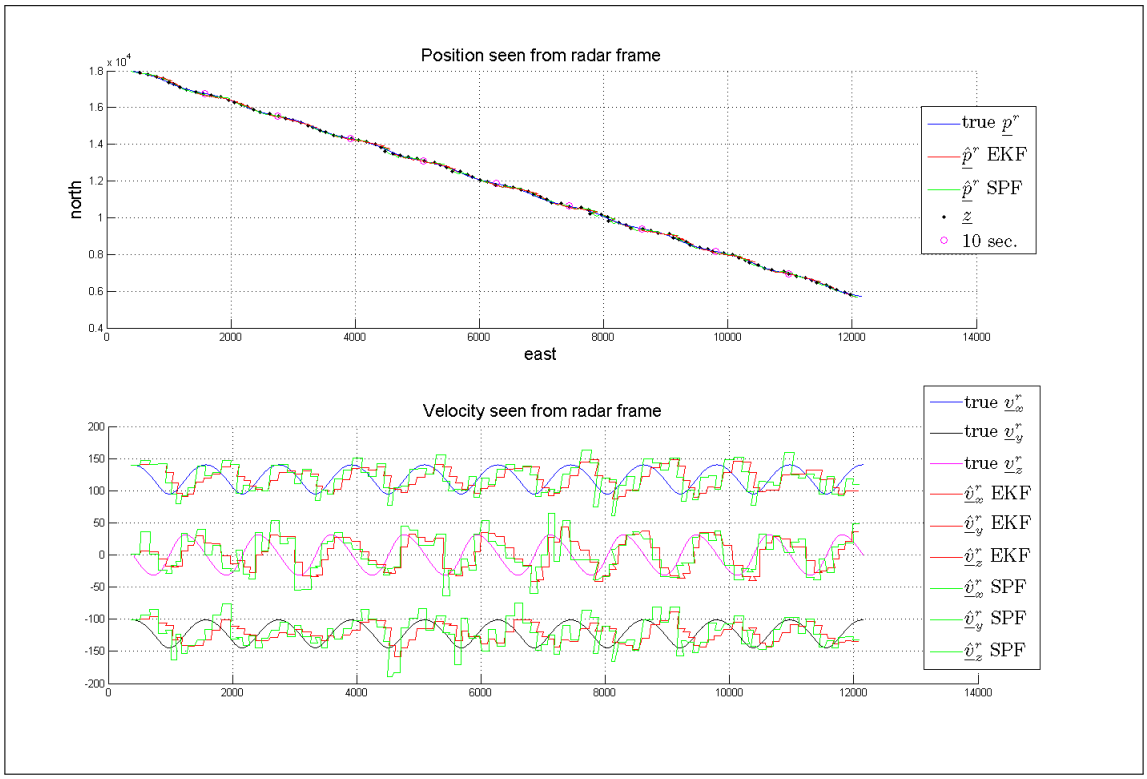

Figure B.10: Scenario 3, helix motion. Airplane position and velocities seen from $\mathcal{F}_r$. Perturbations unknown for the filters.

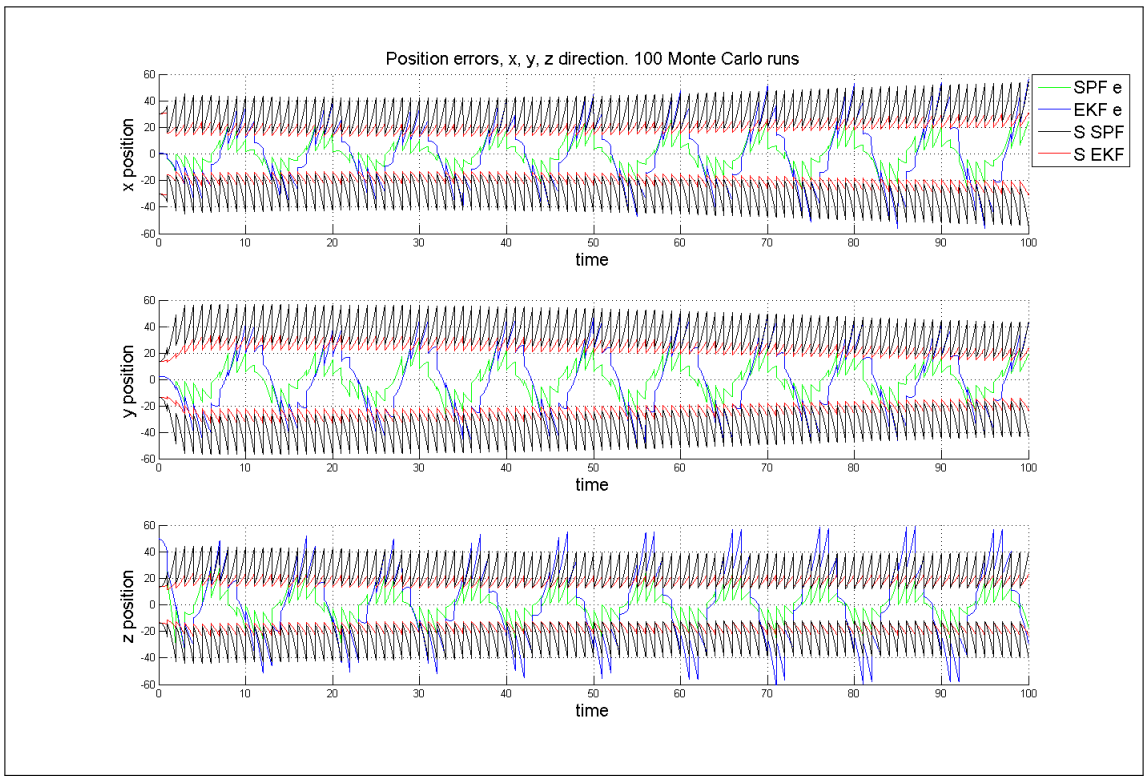

Figure B.11: Scenario 3. Airplane position errors seen from $\mathcal{F}_r$, helix motion. Perturbations unknown for the filters.

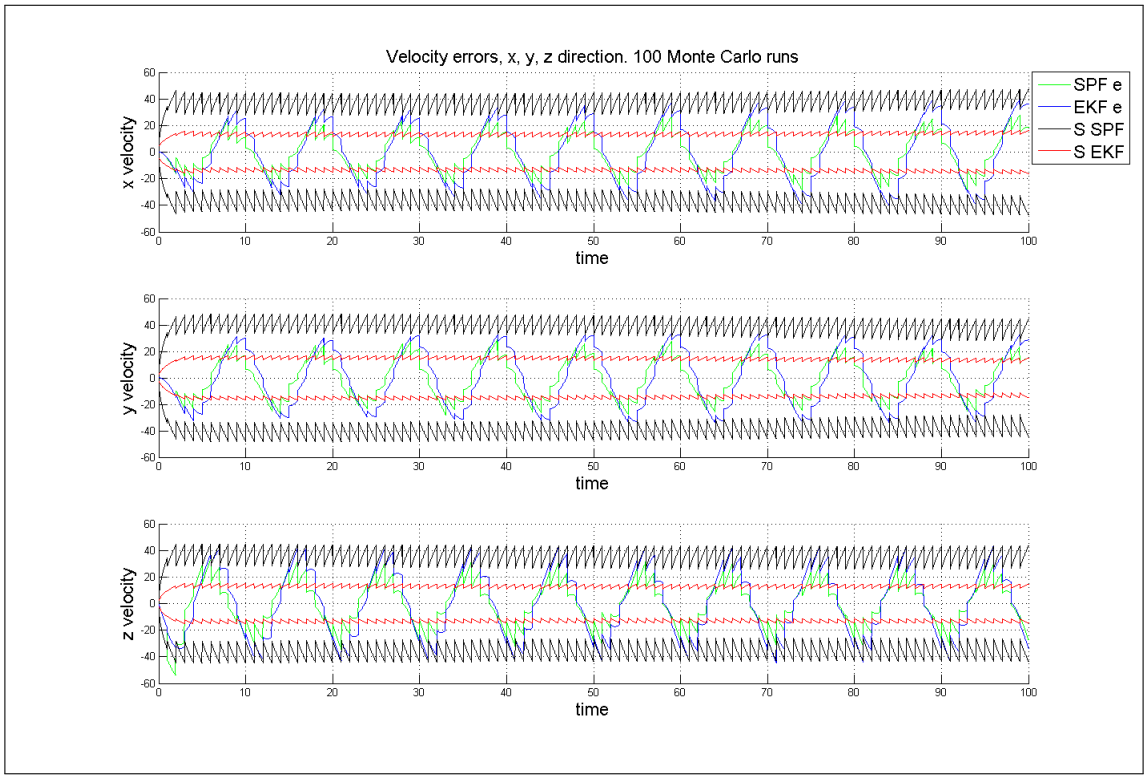

Figure B.12: Scenario 3, helix motion. Airplane velocity errors seen from $\mathcal{F}_r$. Perturbations unknown for the filters.

Appendix C

Program Code

Filters with linear process model, horizontal motion

Main file

```
clear all
close all
format long;

T=0.01;                                 %Discretization time
A=50;                                   %Amplitude
w=2*pi*0.1;                             %Angular velocity
sT = 100;                               %Simulation time in seconds
steps = 10000;
measurement_interval=1/T;               %Measurement update 1Hz
number_of_monte_carlo_runs = 1;

dim=6;                                  % State vector dimension

Rp=[50^2,0,0;0,0.001^2,0;0,0,0.001^2]; %Measurement covariance
Q=eye(3)*0.95^2;
Ga=[zeros(3);eye(3)];
Qm=Ga*Q*Ga';

% Initialization of error trajectories
eXE_EKF=zeros(dim,measurement_interval*sT+sT-1,number_of_monte_carlo_runs);
eXE_SPF=zeros(dim,measurement_interval*sT+sT-1,number_of_monte_carlo_runs);
PE_EKF=zeros(dim,measurement_interval*sT+sT-1,number_of_monte_carlo_runs);
PE_SPF=zeros(dim,measurement_interval*sT+sT-1,number_of_monte_carlo_runs);

for(trajectory=1:number_of_monte_carlo_runs)% Monte Carlo simulations

% Initialization of airplane state vector
vtot=sqrt(170^2+(A*w)^2);
xa=zeros(6,steps);
xa(1,1)=0;
xa(2,1)=0;
xa(3,1)=0;
xa(4,1)=sqrt(vtot^2-(A*w)^2);
xa(5,1)=A*w;
xa(6,1)=0;

% Initialization of airplane state vector seen from the radar
xr=zeros(6,steps);
xr(1,1)=18000;
xr(2,1)=0;
xr(3,1)=100;
```

```
43
44 phi_init=pi+(pi/2)-atan(xr(1,1)/xr(2,1));%+45*pi/180; % Heading angle
45 R_a_r=[cos(phi_init),-sin(phi_init),0;sin(phi_init),cos(phi_init)
       ,0;0,0,1]; %Rotation matrix
46
47 z(1:3,1:steps)=0;
48
49 for t=1:steps-1
50     xa(1,t+1)=xa(1,t)+T/2*(xa(4,t)+xa(4,t));
51     xa(2,t+1)=xa(2,t)+T/2*(xa(5,t)+(xa(5,t)+T*(-w^2*xa(2,t))));
52     xa(3,t+1)=xa(3,t)+T/2*(xa(6,t)+(xa(6,t)));
53     xa(4,t+1)=xa(4,t);
54     xa(5,t+1)=xa(5,t)+T/2*(-w^2*xa(2,t)-w^2*(xa(2,t)+T*xa(5,t)));
55     xa(6,t+1)=xa(6,t);
56 end
57
58 for t=1:steps
59     xr(4:6,t)=R_a_r*xa(4:6,t);
60     xr(1:3,t)=xr(1:3,1)+R_a_r*xa(1:3,t);
61     z(:,t)=[sqrt(xr(1,t)^2+xr(2,t)^2+xr(3,t)^2);atan(xr(2,t)/xr(1,t));acos
           ((xr(3,t))/(sqrt(xr(1,t)^2+xr(2,t)^2+xr(3,t)^2)))]+chol(Rp)*randn
           (3,1);
62 end
63
64
65 %-------------------------------------------------------
66 % Extended Kalman Filter
67 %-------------------------------------------------------
68
69 z0(1:3,1)=z(:,1);
70 x0=z0(1,1)*sin(z0(3,1))*cos(z0(2,1));
71 y0=z0(1,1)*sin(z0(3,1))*sin(z0(2,1));
72 z0=z0(1,1)*cos(z0(3,1));
73 z1(1:3,1)=z(:,100);
74 x1=z1(1,1)*sin(z1(3,1))*cos(z1(2,1));
75 y1=z1(1,1)*sin(z1(3,1))*sin(z1(2,1));
76 z1=z1(1,1)*cos(z1(3,1));
77
78 %Initializing filter state vector
79 x(1:3,1)=[x0;y0;z0];
80 x(4:6,1)=[(x1-x0);(y1-y0);0];
81
82 L=numel(x);
83
84 r=z(1,t);sr=sqrt(Rp(1,1));a=z(2,t);sa=sqrt(Rp(2,2));e=z(3,t);se=sqrt(Rp
       (3,3));
85 R=sphertocartR(r,a,e,sr,sa,se);
86 P=[R,R*T;R*T,2*R*T];
87
88 XE=x;
89 XP=zeros(6,1);
90 PE=P;
91
```

```
92  PED=0;
93  PED(1:L,1)=diag(PE);
94  XED=XE;
95  XEPD=XE;
96
97  F=[zeros(3),eye(3);zeros(3),zeros(3)];
98  Fi=expm(F*T);
99
100 z=z(:,1:length(z));
101 k=0;
102 tk=0;
103 counter_k=0;
104 counter_kE=0;
105 counter_k(1)=1;
106 counter_kE(1)=1;
107 Time=0;
108 TimeE=0;
109 Time(1)=1;
110 TimeE(1)=1;
111
112 %Time update 100Hz
113 %Measurement update 1 Hz
114 for(t=1:steps-1)
115     if(mod(t,1000)==0)
116         tk=tk+1;
117        timeten(:,tk)=xr(:,t);
118     end
119         if(mod(t,100)==0)
120             k=k+1;
121             xyztransform = meastransf(z,Rp,t);
122             xyztrans(:,k)=xyztransform;
123             %Measurement update----------------------------
124             [XE,PE,XED,XEPD,PED,counter_kE] =
                    linmod_horizontal_EKF_measupdate(XP,PP,z,XED,XEPD,PED,
                    counter_kE,Rp,L,t);
125             %Time update----------------------------------
126             [XP,XE,PP,PE,XEPD,PED,counter_k,counter_kE] =
                    linmod_horizontal_EKF_timeupdate(XE,PE,Fi,XEPD,PED,
                    counter_k,counter_kE,Qm);
127         else%Time update----------------------------------
128             [XP,XE,PP,PE,XEPD,PED,counter_k,counter_kE] =
                    linmod_horizontal_EKF_timeupdate(XE,PE,Fi,XEPD,PED,
                    counter_k,counter_kE,Qm);
129         end
130
131 end
132
133 %-------------------------------------------------------
134 % Sigma Point Filter
135 %-------------------------------------------------------
136 %initializations
137 L=size(x,1);
138
```

```
PEu=P;
XEu=x;
XPu=zeros(L,1);

PEuD=0;
PEuD(1:L,1)=diag(PEu);
XEDu=XEu;
XEPDu=XEu;
zpositionu(1,1)=XEDu(1,1);
zpositionu(2,1)=XEDu(3,1);
zpositionu(3,1)=100;

N=2*L+1;
m = size(z,1);
Xa=zeros(L,N);
X=zeros(L,N);

alpha=1;
kappa=0;
beta=2;
lambda=alpha^2*(L+kappa)-L;

Wm=[lambda/(L+lambda)  0.5/(L+lambda)+zeros(1,2*L)];
Wc=Wm;
Wc(1)=Wc(1)+(1-alpha^2+beta);

f = @(x)[x(1,1) + T*x(4,1); x(2,1) + T*x(5,1); x(3,1) + T*x(6,1); x(4,1);
     x(5,1); x(6,1)];% + v(t);% nonlinear state equations
h = @(x)[sqrt(x(1,1)^2+x(2,1)^2+x(3,1)^2);atan(x(2,1)/x(1,1));acos((x
    (3,1))/(sqrt(x(1,1)^2+x(2,1)^2+x(3,1)^2)))];% nonlinear measurement
    equation

k=0;
tk=0;
counter_k=0;
counter_kE=0;
counter_k(1)=1;
counter_kE(1)=1;

%Time update 100Hz
%Measurement update 1 Hz
for(t=1:steps-1)
    if(mod(t,1000)==0)
        tk=tk+1;
        timeten(:,tk)=xr(:,t);
    end
        if(mod(t,100)==0)
            k=k+1;
            xyztransform = meastransf(z,Rp,t);
            xyztrans(:,k)=xyztransform;

            %measurement update
```

```
188                 [XEu,PEu,XEDu,XEPDu,PEuD,counter_kE] =
                        linmod_horizontal_SPF_measupdate(XPu,PPu,XEDu,XEPDu,PEuD,
                        counter_kE,Rp,Wm,Wc,h,X,z,t,m,N);
189                 %Time update----------------------------------------
190                 [XPu,XEu,PPu,PEu,XEPDu,PEuD,counter_k,counter_kE,X] =
                        linmod_horizontal_SPF_timeupdate(XEu,PEu,XEPDu,PEuD,
                        counter_k,counter_kE,f,Qm,Wm,Wc,L,N,lambda);
191         else %Time update----------------------------------------
192                 [XPu,XEu,PPu,PEu,XEPDu,PEuD,counter_k,counter_kE,X] =
                        linmod_horizontal_SPF_timeupdate(XEu,PEu,XEPDu,PEuD,
                        counter_k,counter_kE,f,Qm,Wm,Wc,L,N,lambda);
193             end %time update
194     end %Sigma Point filter
195
196     Time=counter_k.*T;
197     TimeE=counter_kE.*T;
198
199     %Calculate time for estimate
200     TimeEst=0;
201     for(t=1:10:length(Time))
202         if(t==1)
203             TimeEst(t)=T;
204         else
205             TimeEst(end+1)=Time(t-1);
206         end
207     end
208     %calculate time for x true
209     %Tidtrue=[0,Tid];
210
211     %calculate xe time, new xe1.
212     xr1=zeros(dim,length(TimeE));
213     xr1(:,1)=xr(:,1);
214     k=0;
215     for(t=1:length(TimeE))
216         k=k+1;
217         xr1(:,t)=xr(:,k);
218         if(t~=length(TimeE))
219         if(TimeE(t+1)==TimeE(t))
220             xr1(:,t+1)=xr1(:,t);
221             if(t~=length(TimeE))
222                 t=t+1;
223                 k=k-1;
224             end
225         end
226         end
227     end
228
229     %error trajectories
230     eXE_EKF(:,:,trajectory)=xr1-XEPD;
231     eXE_SPF(:,:,trajectory)=xr1-XEPDu;
232     PE_EKF(:,:,trajectory)=PED;
233     PE_SPF(:,:,trajectory)=PEuD;
234     end %Monte Carlo Simulations
```

```
235
236 % Monte Carlo Analysis
237 [mXE_EKF,mXE_SPF,S_EKF_true,S_SPF_true,S_EKF,S_SPF] = MCanalysis_airplane
        (eXE_EKF,eXE_SPF,PE_EKF,PE_SPF,dim,measurement_interval,
        number_of_monte_carlo_runs,sT);
```

EKF time update

```
1  function [XP,XE,PP,PE,XEPD,PED,counter_k,counter_kE] =
       linmod_horizontal_EKF_timeupdate(XE,PE,Fi,XEPD,PED,counter_k,
       counter_kE,Qm)
2
3  % Extended Kalman filter time update
4
5  XP=Fi*XE;
6  PP=Fi*PE*Fi'+Qm;
7
8  XE=XP;
9  PE=PP;
10
11 XEPD(:,end+1)=XE;
12 PED(:,end+1)=diag(PE);
13 counter_k(end+1)=counter_k(end)+1;
14 counter_kE(end+1)=counter_kE(end)+1;
```

EKF measurement update.

```
1  function [XE,PE,XED,XEPD,PED,counter_kE] =
       linmod_horizontal_EKF_measupdate(XP,PP,z,XED,XEPD,PED,counter_kE,Rp,L
       ,t)
2
3  % Extended Kalman filter measurement update
4
5  R1=sqrt(XP(1)^2+XP(2)^2+XP(3)^2);
6  R2=XP(1)^2+XP(2)^2;
7  H=[XP(1)/R1,XP(2)/R1,XP(3)/R1,0,0,0;
8     -XP(2)/R2,XP(1)/R2,0,0,0,0;
9       XP(1)*XP(2)/(R1^2*sqrt(R2)),XP(2)*XP(3)/(R1^2*sqrt(R2)),-sqrt(R2)/R1
            ^2,0,0,0];
10 K=PP*H'/(H*PP*H'+Rp);
11 PE=(eye(L)-K*H)*PP;
12 h=[sqrt(XP(1)^2+XP(2)^2+XP(3)^2);atan(XP(2)/XP(1));acos(XP(3)/(sqrt(XP(1)
       ^2+XP(2)^2+XP(3)^2)))];
13 XE=XP+K*(z(:,t)-h);
14
15 PED(:,end+1)=diag(PE);
16 XED(:,end+1)=XE;
```

```
17 XEPD(:,end+1)=XE;
18 counter_kE(end+1)=counter_kE(end)+1;
```

SPF time update

```
1  function [XPu,XEu,PPu,PEu,XEPDu,PEuD,counter_k,counter_kE,X] =
       linmod_horizontal_SPF_timeupdate(XEu,PEu,XEPDu,PEuD,counter_k,
       counter_kE,f,Qm,Wm,Wc,L,N,lambda)
2
3  % Sigma Point filter time update
4
5  xsigma=chol((L+lambda)*PEu)';
6  Xs=[XEu,XEu*ones(1,N-1)+[xsigma,-xsigma]];
7  XPu=zeros(L,1);
8  for i=1:N
9       X(:,i)=f(Xs(:,i));
10      XPu=XPu+Wm(i)*X(:,i);
11 end
12
13 PPu=zeros(L,L);
14 for(i=1:N)
15     PPu=PPu+Wc(i)*((X(:,i)-XPu)*(X(:,i)-XPu)') + Qm;
16 end
17
18 XEu=XPu;
19 PEu=PPu;
20 PEuD(:,end+1)=diag(PEu);
21 XEPDu(:,end+1)=XEu;
22 counter_k(end+1)=counter_k(end)+1;
23 counter_kE(end+1)=counter_kE(end)+1;
```

SPF measurement update

```
1  function [XEu,PEu,XEDu,XEPDu,PEuD,counter_kE] =
       linmod_horizontal_SPF_measupdate(XPu,PPu,XEDu,XEPDu,PEuD,counter_kE,
       Rp,Wm,Wc,h,X,z,t,m,N)
2
3  %Sigma Point filter measurement update
4
5  zp=zeros(m,1);
6  ZP=zeros(m,N);
7  for i=1:N
8      ZP(:,i)=h(X(:,i));
9      zp=zp+Wm(i)*ZP(:,i);
10 end
11
12 Pzz=(ZP-zp(:,ones(1,N)))*diag(Wc)*(ZP-zp(:,ones(1,N)))'+Rp;
```

```
13  Pxz=(X-XPu(:,ones(1,N)))*diag(Wc)*(ZP-zp(:,ones(1,N)))'; %transformed
        cross-covariance
14  K=Pxz*inv(Pzz);
15  XEu=XPu+K*(z(:,t)-zp);
16  PEu=PPu-K*Pzz*K';
17
18  PEuD(:,end+1)=diag(PEu);
19  XEDu(:,end+1)=XEu;
20  XEPDu(:,end+1)=XEu;
21  counter_kE(end+1)=counter_kE(end);
```

Filters with linear process model, helix motion

Main file

```
clear all
close all
format long;

T=0.01;                                 %Discretization time
A=50;                                   %Amplitude
w=2*pi*0.1;                             %Angular velocity
sT = 100;                               %Simulation time in seconds
steps = 10000;
measurement_interval=1/T;               %Measurement update 1Hz
number_of_monte_carlo_runs = 3;

dim=6;                                  % State vector dimension

Rp=[30^2,0,0;0,0.001^2,0;0,0,0.001^2]; %Measurement covariance
Q=eye(3)*0.95^2;                        %Process noise covariance
Ga=[zeros(3);eye(3)];                   %Process noise matrix
Qm=Ga*Q*Ga';

% Initialization of error trajectories
eXE_EKF=zeros(dim,measurement_interval*sT+sT-1,number_of_monte_carlo_runs);
eXE_SPF=zeros(dim,measurement_interval*sT+sT-1,number_of_monte_carlo_runs);
PE_EKF=zeros(dim,measurement_interval*sT+sT-1,number_of_monte_carlo_runs);
PE_SPF=zeros(dim,measurement_interval*sT+sT-1,number_of_monte_carlo_runs);

for(trajectory=1:number_of_monte_carlo_runs)% Monte Carlo simulations

% Initialization of airplane state vector
vtot=sqrt(170^2+(A*w)^2);
xa=zeros(6,steps);
xa(1,1)=0;
xa(2,1)=0;
xa(3,1)=50;
xa(4,1)=sqrt(vtot^2-(A*w)^2);
xa(5,1)=A*w;
xa(6,1)=0;

% Initialization of airplane state vector seen from the radar
xr=zeros(6,steps);
xr(1,1)=18000;
xr(2,1)=0;
xr(3,1)=50;
```

```
43
44 z(1:3,1:steps)=0; %Measurements initialize
45
46 phi_init=pi+(pi/2)-atan(xr(1,1)/xr(2,1));%+45/180 * pi; % Heading angle
       seen from the radar
47 R_a_r=[cos(phi_init),-sin(phi_init),0;sin(phi_init),cos(phi_init)
       ,0;0,0,1]; %Rotation matrix
48
49 % Simulation of airplane seen from airplane frame.
50 for t=1:steps-1
51     xa(1,t+1)=xa(1,t)+T/2*(xa(4,t)+xa(4,t));
52     xa(2,t+1)=xa(2,t)+T/2*(xa(5,t)+(xa(5,t)+T*(-w^2*xa(2,t))));
53     xa(3,t+1)=xa(3,t)+T/2*(xa(6,t)+(xa(6,t)+T*(-w^2*xa(3,t))));
54     xa(4,t+1)=xa(4,t);
55     xa(5,t+1)=xa(5,t)+T/2*(-w^2*xa(2,t)-w^2*(xa(2,t)+T*xa(5,t)));
56     xa(6,t+1)=xa(6,t)+T/2*(-w^2*xa(3,t)-w^2*(xa(3,t)+T*xa(6,t)));
57 end
58 % Rotate airplane states from airplane frame to radar frame.
59 % Generates measurements.
60 for t=1:steps
61     xr(4:6,t)=R_a_r*xa(4:6,t);
62     xr(1:3,t)=xr(1:3,1)+R_a_r*xa(1:3,t);
63     z(:,t)=[sqrt(xr(1,t)^2+xr(2,t)^2+xr(3,t)^2);atan(xr(2,t)/xr(1,t));acos
           ((xr(3,t))/(sqrt(xr(1,t)^2+xr(2,t)^2+xr(3,t)^2)))]+chol(Rp)*randn
           (3,1);
64 end
65
66
67 %-------------------------------------------------------
68 % Extended Kalman Filter
69 %-------------------------------------------------------
70
71 %Initializing filter state vector
72 z0(1:3,1)=z(:,1);
73 x0=z0(1,1)*sin(z0(3,1))*cos(z0(2,1));
74 y0=z0(1,1)*sin(z0(3,1))*sin(z0(2,1));
75 z0=z0(1,1)*cos(z0(3,1));
76 z1(1:3,1)=z(:,100);
77 x1=z1(1,1)*sin(z1(3,1))*cos(z1(2,1));
78 y1=z1(1,1)*sin(z1(3,1))*sin(z1(2,1));
79 z1=z1(1,1)*cos(z1(3,1));
80
81 %Initializing filter state vector
82 x(1:3,1)=[x0;y0;z0];
83 x(4:6,1)=[(x1-x0);(y1-y0);(z1-z0)];
84 L=numel(x);
85
86 %Calculate measurement covariance matrix to cartesian coordinates.
87 r=z(1,t);sr=sqrt(Rp(1,1));a=z(2,t);sa=sqrt(Rp(2,2));e=z(3,t);se=sqrt(Rp
       (3,3));
88 R=sphertocartR(r,a,e,sr,sa,se);
89 P=[R,R*T;R*T,2*R*T]; %Initialize covariance matrix.
90
```

```
91  XE=x;
92  XP=zeros(6,1);
93  PE=P;
94
95  PED=0;
96  PED(1:L,1)=diag(PE);
97  XED=XE;
98  XEPD=XE;
99
100 F=[zeros(3),eye(3);zeros(3),zeros(3)];
101 Fi=expm(F*T);
102
103 %z=z(:,1:length(z));
104 k=0;
105 tk=0;
106 counter_k=0;
107 counter_kE=0;
108 counter_k(1)=1;
109 counter_kE(1)=1;
110 Time=0;
111 TimeE=0;
112 Time(1)=1;
113 TimeE(1)=1;
114 %Time update 100Hz
115 %Measurement update 1 Hz
116 for t=1:steps-1
117     if(mod(t,1000)==0)
118         tk=tk+1;
119        timeten(:,tk)=xr(:,t);
120     end
121         if(mod(t,100)==0)
122             k=k+1;
123             xyztransform = meastransf(z,Rp,t);
124             xyztrans(:,k)=xyztransform;
125             %Measurement update----------------------------
126             [XE,PE,XED,XEPD,PED,counter_kE] = linmod_helix_EKF_measupdate
                    (XP,PP,z,XED,XEPD,PED,counter_kE,Rp,L,t);
127             %Time update----------------------------------
128             [XP,XE,PP,PE,XEPD,PED,counter_k,counter_kE] =
                    linmod_helix_EKF_timeupdate(XE,PE,Fi,XEPD,PED,counter_k,
                    counter_kE,Qm);
129         else%Time update----------------------------------
130             [XP,XE,PP,PE,XEPD,PED,counter_k,counter_kE] =
                    linmod_helix_EKF_timeupdate(XE,PE,Fi,XEPD,PED,counter_k,
                    counter_kE,Qm);
131         end
132
133 end
134
135 %-------------------------------------------------------
136 % Sigma Point Filter
137 %-------------------------------------------------------
138 %initializations
```

```
139 L=size(x,1);
140
141 PEu=P;
142 XEu=x;
143 XPu=zeros(L,1);
144
145 PEuD=0;
146 PEuD(1:L,1)=diag(PEu);
147 XEDu=XEu;
148 XEPDu=XEu;
149
150 N=2*L+1;
151 m = size(z,1);
152 Xa=zeros(L,N);
153 X=zeros(L,N);
154
155 alpha=1;
156 kappa=0;
157 beta=2;
158 lambda=alpha^2*(L+kappa)-L;
159
160 Wm=[lambda/(L+lambda) 0.5/(L+lambda)+zeros(1,2*L)];
161 Wc=Wm;
162 Wc(1)=Wc(1)+(1-alpha^2+beta);
163
164 f = @(x)[x(1,1) + T*x(4,1); x(2,1) + T*x(5,1); x(3,1) + T*x(6,1); x(4,1);
      x(5,1); x(6,1)];
165 h = @(x)[sqrt(x(1,1)^2+x(2,1)^2+x(3,1)^2);atan(x(2,1)/x(1,1));acos((x
    (3,1))/(sqrt(x(1,1)^2+x(2,1)^2+x(3,1)^2)))];
166
167 k=0;
168 tk=0;
169 counter_k=0;
170 counter_kE=0;
171 counter_k(1)=1;
172 counter_kE(1)=1;
173 %Time update 100Hz
174 %Measurement update 1 Hz
175 for(t=1:steps-1)
176     if(mod(t,1000)==0)
177         tk=tk+1;
178         timeten(:,tk)=xr(:,t);
179     end
180         if(mod(t,100)==0) %measurement update
181             k=k+1;
182             xyztransform = meastransf(z,Rp,t);
183             xyztrans(:,k)=xyztransform;
184
185             [XEu,PEu,XEDu,XEPDu,PEuD,counter_kE] =
                    linmod_helix_SPF_measupdate(XPu,PPu,XEDu,XEPDu,PEuD,
                    counter_kE,Rp,Wm,Wc,h,X,z,t,m,N);
186             %Time update----------------------------------
187             [XPu,XEu,PPu,PEu,XEPDu,PEuD,counter_k,counter_kE,X] = ...
```

```
188                     linmod_helix_SPF_timeupdate(XEu,PEu,XEPDu,PEuD,counter_k,
                            counter_kE,f,Qm,Wm,Wc,L,N,lambda);
189         else %Time update------------------------------------------
190
191                 [XPu,XEu,PPu,PEu,XEPDu,PEuD,counter_k,counter_kE,X] = ...
192                     linmod_helix_SPF_timeupdate(XEu,PEu,XEPDu,PEuD,counter_k,
                            counter_kE,f,Qm,Wm,Wc,L,N,lambda);
193         end %time update
194 end %Sigma Point filter
195
196 Time=counter_k.*T;
197 TimeE=counter_kE.*T;
198
199 %Calculate time for estimate
200 TimeEst=0;
201 for(t=1:10:length(Time))
202     if(t==1)
203         TimeEst(t)=T;
204     else
205         TimeEst(end+1)=Time(t-1);
206     end
207 end
208 %calculate time for x true
209 %Tidtrue=[0,Tid];
210
211 %calculate xe time, new xe1.
212 xr1=zeros(dim,length(TimeE));
213 xr1(:,1)=xr(:,1);
214 k=0;
215 for(t=1:length(TimeE))
216     k=k+1;
217     xr1(:,t)=xr(:,k);
218     if(t~=length(TimeE))
219     if(TimeE(t+1)==TimeE(t))
220         xr1(:,t+1)=xr1(:,t);
221         if(t~=length(TimeE))
222             t=t+1;
223             k=k-1;
224         end
225     end
226     end
227 end
228
229 %error trajectories
230 eXE_EKF(:,:,trajectory)=xr1-XEPD;
231 eXE_SPF(:,:,trajectory)=xr1-XEPDu;
232 PE_EKF(:,:,trajectory)=PED;
233 PE_SPF(:,:,trajectory)=PEuD;
234 end %Monte Carlo Simulations
235
236 %Monte Carlo analysis
237 [mXE_EKF,mXE_SPF,S_EKF_true,S_SPF_true,S_EKF,S_SPF] = MCanalysis_airplane
        (eXE_EKF,eXE_SPF,PE_EKF,PE_SPF,dim,measurement_interval,
```

```
    number_of_monte_carlo_runs,sT);
```

EKF time update.

```
function [XP,XE,PP,PE,XEPD,PED,counter_k,counter_kE] =
    linmod_helix_EKF_timeupdate(XE,PE,Fi,XEPD,PED,counter_k,counter_kE,Qm
    )

%Extended Kalman filter time update

XP=Fi*XE;
PP=Fi*PE*Fi'+Qm;

XE=XP;
PE=PP;

PED(:,end+1)=diag(PE);
XEPD(:,end+1)=XE;
counter_k(end+1)=counter_k(end)+1;
counter_kE(end+1)=counter_kE(end)+1;
```

EKF measurement update

```
function [XE,PE,XED,XEPD,PED,counter_kE] = linmod_helix_EKF_measupdate(XP
    ,PP,z,XED,XEPD,PED,counter_kE,Rp,L,t)

%Extended Kalman filter measurement update

R1=sqrt(XP(1)^2+XP(2)^2+XP(3)^2);
R2=XP(1)^2+XP(2)^2;
H=[XP(1)/R1,XP(2)/R1,XP(3)/R1,0,0,0;
    -XP(2)/R2,XP(1)/R2,0,0,0,0;
    XP(1)*XP(2)/(R1^2*sqrt(R2)),XP(2)*XP(3)/(R1^2*sqrt(R2)),-sqrt(R2)/R1
        ^2,0,0,0];
K=PP*H'/(H*PP*H'+Rp);
PE=(eye(L)-K*H)*PP;
h=[sqrt(XP(1)^2+XP(2)^2+XP(3)^2);atan(XP(2)/XP(1));acos(XP(3)/(sqrt(XP(1)
    ^2+XP(2)^2+XP(3)^2)))];
XE=XP+K*(z(:,t)-h);


PED(:,end+1)=diag(PE);
XED(:,end+1)=XE;
XEPD(:,end+1)=XE;
counter_kE(end+1)=counter_kE(end)+1;
```

SPF time update

```
function [XPu,XEu,PPu,PEu,XEPDu,PEuD,counter_k,counter_kE,X] = ...
    linmod_helix_SPF_timeupdate(XEu,PEu,XEPDu,PEuD,counter_k,counter_kE,f
        ,Qm,Wm,Wc,L,N,lambda)

% Sigma Point filter time update

X=zeros(L,N);
xsigma=chol((L+lambda)*PEu)';
Xs=[XEu,XEu*ones(1,N-1)+[xsigma,-xsigma]];
XPu=zeros(L,1);
for i=1:N
    X(:,i)=f(Xs(:,i));
    XPu=XPu+Wm(i)*X(:,i);
end

PPu=zeros(L,L);
for i=1:N
    PPu=PPu+Wc(i)*((X(:,i)-XPu)*(X(:,i)-XPu)') + Qm;
end

XEu=XPu;
PEu=PPu;
PEuD(:,end+1)=diag(PEu);
XEPDu(:,end+1)=XEu;
counter_k(end+1)=counter_k(end)+1;
counter_kE(end+1)=counter_kE(end)+1;
```

SPF measurement update

```
function [XEu,PEu,XEDu,XEPDu,PEuD,counter_kE] =
    linmod_helix_SPF_measupdate(XPu,PPu,XEDu,XEPDu,PEuD,counter_kE,Rp,Wm,
    Wc,h,X,z,t,m,N)

% Sigma Point filter measurement update

zp=zeros(m,1);
ZP=zeros(m,N);
for i=1:N
    ZP(:,i)=h(X(:,i));
    zp=zp+Wm(i)*ZP(:,i);
end

Pzz=(ZP-zp(:,ones(1,N)))*diag(Wc)*(ZP-zp(:,ones(1,N)))'+Rp;
Pxz=(X-XPu(:,ones(1,N)))*diag(Wc)*(ZP-zp(:,ones(1,N)))';
K=Pxz*inv(Pzz);
XEu=XPu+K*(z(:,t)-zp);
PEu=PPu-K*Pzz*K';
```

```
17
18 PEuD(:,end+1)=diag(PEu);
19 XEDu(:,end+1)=XEu;
20 XEPDu(:,end+1)=XEu;
21 counter_kE(end+1)=counter_kE(end);
```

Filters with non-linear process model, horizontal wave motion

Main file

```
clear all
close all
format long;
T=0.01;

A=50;
w=2*pi*0.1;
sT = 80;
steps=10000;
measurement_interval=1/T ;

number_of_monte_carlo_runs = 3;
dim=11;
eXE_EKF=zeros(dim,measurement_interval*sT+sT-1,number_of_monte_carlo_runs);
eXE_SPF=zeros(dim,measurement_interval*sT+sT-1,number_of_monte_carlo_runs);
PE_EKF=zeros(dim,measurement_interval*sT+sT-1,number_of_monte_carlo_runs);
PE_SPF=zeros(dim,measurement_interval*sT+sT-1,number_of_monte_carlo_runs);

for(trajectory=1:number_of_monte_carlo_runs)

vtot=sqrt(170^2+(A*w)^2);
xa=zeros(7,steps);
xa(1,1)=0;
xa(2,1)=0;
xa(3,1)=0;
xa(4,1)=sqrt(vtot^2-(A*w)^2);
xa(5,1)=A*w;
xa(6,1)=0;
xa(7,1)=w;

xr=zeros(11,steps);
xr(1,1)=18000;
xr(2,1)=0;
xr(3,1)=100;

coursee=zeros(3,steps);
coursee(1:3,1)=xr(1:3,1);
xr(8:10,1)=coursee(1:3,1);
phi_init=pi+(pi/2)-atan(xr(1,1)/xr(2,1));%+45*pi/180;
R_a_r=[cos(phi_init),-sin(phi_init),0;sin(phi_init),cos(phi_init),0;0,0,1]; % rotation matrix
```

```
41  xr(4:6,1)=R_a_r*xr(4:6,1);
42
43  Rp=[50^2,0,0;0,0.001^2,0;0,0,0.001^2];
44
45  Q=eye(3)*0.05^2;
46  Ga=[zeros(3);eye(3);eye(3);[0,0,0;0,0,0]];
47  Qm=Ga*Q*Ga';
48  z(1:3,1:steps)=0;
49  z(1:3,1)=[sqrt(xr(1,1)^2+xr(2,1)^2+xr(3,1)^2);atan(xr(2,1)/xr(1,1));acos
        ((xr(3,1))/(sqrt(xr(1,1)^2+xr(2,1)^2+xr(3,1)^2)))];%;+chol(R)*randn
        (3,1);
50  za(1:3,1)=0;
51
52  for t=1:steps-1
53      xa(1,t+1)=xa(1,t)+T/2*(xa(4,t)+xa(4,t));
54      xa(2,t+1)=xa(2,t)+T/2*(xa(5,t)+(xa(5,t)+T*(-w^2*xa(2,t))));
55      xa(3,t+1)=xa(3,t)+T/2*(xa(6,t)+xa(6,t));
56      xa(4,t+1)=xa(4,t);
57      xa(5,t+1)=xa(5,t)+T/2*(-w^2*xa(2,t)-w^2*(xa(2,t)+T*xa(5,t)));
58      xa(6,t+1)=xa(6,t);
59      xa(7,t+1)=xa(7,t);
60  end
61  p_ra_r=0;
62  for t=1:steps
63      xr(4:6,t)=R_a_r*xa(4:6,t);
64      xr(1:3,t)=xr(1:3,1)+R_a_r*xa(1:3,t);
65      if(t==2000)
66      p_ra_r=xr(1:3,t);
67      end
68      xr(7:9,t)=p_ra_r;
69      xr(10,t)=phi_init;
70      xr(11,t)=w;
71
72      z(:,t)=[sqrt(xr(1,t)^2+xr(2,t)^2+xr(3,t)^2);atan(xr(2,t)/xr(1,t));acos
            ((xr(3,t))/(sqrt(xr(1,t)^2+xr(2,t)^2+xr(3,t)^2)))]+chol(Rp)*randn
            (3,1);
73  end
74
75
76  %-------------------------------------------------------------
77  % Extended Kalman Filter
78  %-------------------------------------------------------------
79
80  R3 = @(x)[cos(x),-sin(x),0;sin(x),cos(x),0;0,0,1];
81
82  counter_k=0;
83  counter_kE=0;
84  counter_k(1)=1;
85  counter_kE(1)=1;
86  kpre=0;
87  k=0;
88  tk=0;
89
```

```
90  xyztransa=0;
91  xyztransa(1:3,1)=0;
92  xyztrans=0;
93  xyztrans(1:3,1)=0;
94  X_freq_est_ekf_pre=0;
95  X_freq_est_ekf_pre(1:3,1)=0;
96  c(1:3,1)=0;
97  course_xyz(1:3,1)=0;
98  XE(1:11,1)=0;
99  XED=XE;
100 XEPD=XE;
101 XEDr=XE;
102 XEPDr=XE;
103
104 for(t=1:steps-1)
105     if(mod(t,1000)==0)
106         tk=tk+1;
107         timeten(:,tk)=xr(:,t);
108     end
109
110     if t≤2000
111         if(mod(t,measurement_interval)==0)%Pre-Measurement update
112             kpre=kpre+1;
113             xyztransform = meastransf(z,Rp,t);
114             xyztrans(:,kpre)=xyztransform;
115
116             c(:,kpre)=xyztrans(:,kpre);
117             [course_x,course_y,course_z] = courseest(c,kpre);
118             course_xyz(1:3,kpre)=[course_x;course_y;course_z];
119
120             X_freq_est_ekf_pre(:,kpre)=xyztrans(:,kpre);
121             if(mod(kpre,10)==0)
122                 freq=music(X_freq_est_ekf_pre,T,measurement_interval);
123                 XE(11,1)=freq;
124             end
125
126             if(kpre==20)
127
128             if(course_xyz(1,end)>course_xyz(1,1))
129                 phi=3*pi/2 + atan((course_xyz(2,1)-course_xyz(2,end))/(
                        course_xyz(1,end)-course_xyz(1,1)));
130             else
131                 phi=pi+atan((course_xyz(2,1)-course_xyz(2,end))/(course_xyz
                        (1,1)-course_xyz(1,end)));
132             end
133             for i=1:kpre
134                 xyztransa(:,i)=R3(phi)'*xyztrans(:,i);
135             end
136
137
138             p0r=course_xyz(:,end);
139
140             XE(1:3,1)=[0;0;0];
```

```
141             xvelocity=(xyztransa(1,end)-xyztransa(1,1))./length(xyztransa);
142             [yvelocitymin,indexmin]=min(xyztransa(2,:));
143             [yvelocitymax,indexmax]=max(xyztransa(2,:));
144             yvelocity=(abs(yvelocitymax-yvelocitymin))/(abs(indexmin-
                    indexmax));
145
146             XE(4:6,1)=[xvelocity;yvelocity;0];
147             XE(7:9,1)=p0r;
148             XE(10,1)=phi;
149             XP=XE;
150
151             XED(1:length(XE),1)=XE;
152             XEPD(1:length(XE),1)=XE;
153             XEPDr(1:length(XE),1)=XE;
154             XEDr(1:length(XE),1)=XE;
155             XEPDr(:,1)=[[p0r+R3(phi)*XE(1:3,1)];[R3(phi)*XE(4:6,1)];XE
                    (7:11,1)];
156             XEDr(:,1)=[[p0r+R3(phi)*XE(1:3,1)];[R3(phi)*XE(4:6,1)];XE
                    (7:11,1)];
157
158             zposition=0;
159             zposition(1,1)=XED(1,1);
160             zposition(2,1)=XED(3,1);
161             zposition(3,1)=100;
162             L=numel(XE);
163
164             r=z(1,t);sr=sqrt(Rp(1,1));a=z(2,t);sa=sqrt(Rp(2,2));e=z(3,t);se
                    =sqrt(Rp(3,3));
165             R=sphertocartR(r,a,e,sr,sa,se);
166
167             PE=zeros(L);
168             PE(1:9,1:9)=[R,zeros(3),zeros(3);zeros(3),R,zeros(3);zeros(3),
                    zeros(3),R];
169             PE(10,10)=0.1;
170             PE(11,11)=0.1;
171             PP=PE;
172             PED=0;
173             PED(1:L,1)=diag(PE);
174             end
175         end
176     else %start estimator
177     if(mod(t,measurement_interval)==0)
178               k=k+1;
179               kpre=kpre+1;
180
181               xyztransform = meastransf(z,Rp,t);
182               xyztrans(:,kpre)=xyztransform;
183               %Measurement update----------------------------
184               X_freq_est_ekf_pre(:,kpre)=xyztrans(:,kpre);
185               freq=music(X_freq_est_ekf_pre,T,measurement_interval);
186               XE(11,1)=freq;
187               XP(11,1)=freq;
188
```

```
189                 %calculate measurement covariance
190                 r=z(1,t);sr=sqrt(Rp(1,1));a=z(2,t);sa=sqrt(Rp(2,2));e=z(3,t);
                        se=sqrt(Rp(3,3));
191                 R=sphertocartR(r,a,e,sr,sa,se);
192
193                 %Measurement update-----------------------------
194                 [XE,XP,PE,PP,XED,XEPD,XEDr,XEPDr,PED,zposition,counter_kE] =
                        ...
195                     horizontal_ekf_measupdate(XE,XP,PP,XED,XEPD,XEDr,XEPDr,
                            PED,xyztrans,freq,counter_kE,zposition,R3,R,L,kpre,k)
                            ;
196
197                 %Time update-----------------------------------
198                 [XE,XP,PE,PP,XEPD,XEPDr,PED,counter_k,counter_kE]=...
199                     horizontal_ekf_timeupdate(XE,PE,XEPD,XEPDr,PED,counter_k,
                            counter_kE,R3,T,Qm);
200
201         else %Time update
202             [XE,XP,PE,PP,XEPD,XEPDr,PED,counter_k,counter_kE]=...
203                 horizontal_ekf_timeupdate(XE,PE,XEPD,XEPDr,PED,counter_k,
                        counter_kE,R3,T,Qm);
204         end
205
206         end
207 end %Extended kalman filter
208
209 %-------------------------------------------------------------
210 % Sigma Point Filter
211 %-------------------------------------------------------------
212
213 L=11;
214 N=2*L+1;
215 m = size(z,1);
216 Xa=zeros(L,N);
217 X=zeros(L,N);
218
219 alpha=0.001;
220 kappa=0;
221 beta=2;
222 lambda=alpha^2*(L+kappa)-L;
223
224 Wm=[lambda/(L+lambda)  0.5/(L+lambda)+zeros(1,2*L)];
225 Wc=Wm;
226 Wc(1)=Wc(1)+(1-alpha^2+beta);
227
228 f = @(x)[x(1,1) + T*x(4,1); x(2,1) + T*x(5,1); x(3,1) + T*x(6,1); x(4,1);
        x(5,1) - T*x(11,1)^2*x(2,1); x(6,1); x(7,1);x(8,1);x(9,1);x(10,1);x
        (11,1)];
229 h = @(x)[[x(7,1);x(8,1);x(9,1)] + [cos(x(10,1)),-sin(x(10,1)),0;sin(x
        (10,1)),cos(x(10,1)),0;0,0,1]*[x(1,1);x(2,1);x(3,1)]];
230 R3 = @(x)[cos(x),-sin(x),0;sin(x),cos(x),0;0,0,1];
231
232 counter_k=0;
```

```
233 counter_kE=0;
234 counter_k(1)=1;
235 counter_kE(1)=1;
236 kpre=0;
237 k=0;
238 tk=0;
239 tell=0;
240 timeup=0;
241
242 XEu(1:L,1)=0;
243 xyztransa=0;
244 xyztransa(1:3,1)=0;
245 xyztrans=0;
246 xyztrans(1:3,1)=0;
247 X_freq_est_spf_pre=0;
248 X_freq_est_spf_pre(1:3,1)=0;
249 c(1:3,1)=0;
250 course_xyz(1:3,1)=0;
251
252 XEDu=XEu;
253 XEPDu=XEu;
254 XEDur=XEu;
255 XEPDur=XEu;
256
257 for(t=1:steps-1)
258
259     if(mod(t,1000)==0)
260         tk=tk+1;
261         timeten(:,tk)=xr(:,t);
262     end
263
264     if t≤2000
265         if(mod(t,measurement_interval)==0)%Pre-Measurement update
266             kpre=kpre+1;
267             xyztransform = meastransf(z,Rp,t);
268             xyztrans(:,kpre)=xyztransform;
269
270             c(:,kpre)=xyztrans(:,kpre);
271             [course_x,course_y,course_z] = courseest(c,kpre);
272             course_xyz(1:3,kpre)=[course_x;course_y;course_z];
273
274             X_freq_est_spf_pre(:,kpre)=xyztrans(:,kpre);
275             if(mod(kpre,10)==0)
276                 freq=music(X_freq_est_spf_pre,T,measurement_interval);
277                 XEu(11,1)=freq;
278             end
279
280             if(kpre==20)
281             if(course_xyz(1,end)>course_xyz(1,1))
282                 phi=3*pi/2 + atan((course_xyz(2,1)-course_xyz(2,end))/(
                        course_xyz(1,end)-course_xyz(1,1)))
283             else
```

```
284                 phi=pi+atan((course_xyz(2,1)-course_xyz(2,end))/(course_xyz
                        (1,1)-course_xyz(1,end)))
285             end
286             for i=1:kpre
287             xyztransa(:,i)=R3(phi)'*xyztrans(:,i);
288             end
289             p0r=course_xyz(:,end);
290 
291             XEu(1:3,1)=[0;0;0];
292             xvelocity=(xyztransa(1,end)-xyztransa(1,1))./length(xyztransa);
293             [yvelocitymin,indexmin]=min(xyztransa(2,:));
294             [yvelocitymax,indexmax]=max(xyztransa(2,:));
295             yvelocity=(abs(yvelocitymax-yvelocitymin))/(abs(indexmin-
                    indexmax));
296             XEu(4:6,1)=[xvelocity;yvelocity;0];
297             XEu(7:9,1)=p0r;
298             XEu(10,1)=phi;
299             XPu=XEu;
300 
301             XEDu(1:length(XEu),1)=XEu;
302             XEPDu(1:length(XEu),1)=XEu;
303 
304             XEPDur(1:length(XEu),1)=XEu;
305             XEDur(1:length(XEu),1)=XEu;
306             XEPDur(:,1)=[[p0r+R3(phi)*XEu(1:3,1)];[R3(phi)*XEu(4:6,1)];XEu
                    (7:11,1)];
307             XEDur(:,1)=[[p0r+R3(phi)*XEu(1:3,1)];[R3(phi)*XEu(4:6,1)];XEu
                    (7:11,1)];
308 
309             zposition=0;
310             zposition(1,1)=XEDu(1,1);
311             zposition(2,1)=XEDu(3,1);
312             zposition(3,1)=100;
313             L=numel(XEu);
314 
315             r=z(1,t);sr=sqrt(Rp(1,1));a=z(2,t);sa=sqrt(Rp(2,2));e=z(3,t);se
                    =sqrt(Rp(3,3));
316             R=sphertocartR(r,a,e,sr,sa,se);
317 
318             PEu=zeros(L);
319             PEu(1:9,1:9)=[R,zeros(3),zeros(3);zeros(3),R,zeros(3);zeros(3),
                    zeros(3),R];
320             PEu(10,10)=0.1;
321             PEu(11,11)=0.1;
322             PPu=PEu;
323             PEuD=0;
324             PEuD(1:L,1)=diag(PEu);
325             end
326         end
327     else %start estimator
328         if(mod(t,measurement_interval)==0) %measurement update
329               k=k+1;
330               kpre=kpre+1;
```

```
331                 xyztransform = meastransf(z,Rp,t);
332                 xyztrans(:,kpre)=xyztransform;
333
334                 r=z(1,t);sr=sqrt(Rp(1,1));a=z(2,t);sa=sqrt(Rp(2,2));e=z(3,t);
                        se=sqrt(Rp(3,3));
335                 R=sphertocartR(r,a,e,sr,sa,se);
336
337                 X_freq_est_spf_pre(:,kpre)=xyztrans(:,kpre);
338                 freq=music(X_freq_est_spf_pre,T,measurement_interval);
339                 XEu(11,1)=freq;
340                 XPu(11,1)=freq;
341
342                 %Measurement update----------------------------
343                 [XEu,XPu,PEu,PPu,XEDu,XEPDu,XEDur,XEPDur,PEuD,X,counter_kE]
                        =...
344                  horizontal_spf_measupdate(XPu,XEu,PPu,PEu,XEDu,XEPDu,
                         XEDur,XEPDur,PEuD,Wc,Wm,xyztrans,freq,counter_kE,N,h,
                         f,R3,R,m,L,lambda,kpre,X);
345
346                 %Time update----------------------------------
347                 [XEu,XPu,PEu,PPu,X,XEPDu,XEPDur,PEuD,counter_k,counter_kE]=
                        horizontal_spf_timeupdate...
348                  (XEu,XPu,PEu,PPu,XEPDu,XEPDur,PEuD,Wc,Wm,counter_k,
                         counter_kE,Q,R3,f,L,lambda,N,Qm);
349
350       else %Time update----------------------------------------
351             [XEu,XPu,PEu,PPu,X,XEPDu,XEPDur,PEuD,counter_k,counter_kE]=
                    horizontal_spf_timeupdate...
352                  (XEu,XPu,PEu,PPu,XEPDu,XEPDur,PEuD,Wc,Wm,counter_k,counter_kE
                         ,Q,R3,f,L,lambda,N,Qm);
353
354          end
355       end
356  end
357
358  Time=counter_k.*T;
359  TimeE=counter_kE.*T;
360
361  %Calculate time for estimate
362  TimeEst=0;
363  for(t=1:10:length(Time))
364      if(t==1)
365          TimeEst(t)=T;
366      else
367          TimeEst(end+1)=Time(t-1);
368      end
369  end
370
371  %calculate xe time, new xe1.
372  prem=20;
373  xr1=zeros(dim,length(TimeE));
374  xrnew=xr(:,2000:end);
375  xr1(:,1)=xr(:,2000);
```

```
376 k=0;
377 for(t=1:length(TimeE))
378     k=k+1;
379     xr1(:,t)=xrnew(:,k);
380     if(t≠length(TimeE))
381     if(TimeE(t+1)==TimeE(t))
382         xr1(:,t+1)=xr1(:,t);
383         if(t≠length(TimeE))
384             t=t+1;
385             k=k-1;
386         end
387     end
388     end
389 end
390
391 %error trajectories
392 eXE_EKF(:,:,trajectory)=xr1-XEPDr;
393 eXE_SPF(:,:,trajectory)=xr1-XEPDur;
394 PE_EKF(:,:,trajectory)=PED;
395 PE_SPF(:,:,trajectory)=PEuD;
396 end %Monte Carlo Simulations
397
398 %Monte Carlo Analysis
399 [mXE_EKF,mXE_SPF,S_EKF_true,S_SPF_true,S_EKF,S_SPF] = MCanalysis_airplane
      (eXE_EKF,eXE_SPF,PE_EKF,PE_SPF,dim,measurement_interval,
      number_of_monte_carlo_runs,sT);
```

EKF time update.

```
1  function [XE,XP,PE,PP,XEPD,XEPDr,PED,counter_k,counter_kE]=
       horizontal_ekf_timeupdate(XE,PE,XEPD,XEPDr,PED,counter_k,counter_kE,
       R3,T,Qm)
2
3  % Extended Kalman filter time update
4
5  XP(1,1)=XE(1)+T*XE(4);
6  XP(2,1)=XE(2)+T*XE(5);
7  XP(3,1)=XE(3)+T*XE(6);
8  XP(4,1)=XE(4);
9  XP(5,1)=XE(5)-XE(11)^2*XE(2)*T;
10 XP(6,1)=XE(6);
11 XP(7,1)=XE(7);
12 XP(8,1)=XE(8);
13 XP(9,1)=XE(9);
14 XP(10,1)=XE(10);
15 XP(11,1)=XE(11);
16
17 F=[0,0,0,1,0,0,0,0,0,0,0;
18     0,0,0,0,1,0,0,0,0,0,0;
19     0,0,0,0,0,1,0,0,0,0,0;
```

```
      0,0,0,0,0,0,0,0,0,0,0;
      0,-XE(11,1)^2,0,0,0,0,0,0,0,0,-2*XE(11,1)*XE(2,1);
      0,0,0,0,0,0,0,0,0,0,0;
      0,0,0,0,0,0,0,0,0,0,0;
      0,0,0,0,0,0,0,0,0,0,0;
      0,0,0,0,0,0,0,0,0,0,0;
      0,0,0,0,0,0,0,0,0,0,0;
      0,0,0,0,0,0,0,0,0,0,0];

Fi=expm(F*T);
PP=Fi*PE*Fi'+Qm;

XE=XP;
PE=PP;
XEPD(:,end+1)=XE;
XEPDr(:,end+1)=[[XE(7:9,1)+R3(XE(10,1))*XE(1:3,1)];[R3(XE(10,1))*XE
    (4:6,1)];XE(7:11,1)];
PED(:,end+1)=diag(PE);
counter_k(end+1)=counter_k(end)+1;
counter_kE(end+1)=counter_kE(end)+1;
```

EKF measurement update

```
function [XE,XP,PE,PP,XED,XEPD,XEDr,XEPDr,PED,zposition,counter_kE] =
    horizontal_ekf_measupdate(XE,XP,PP,XED,XEPD,XEDr,XEPDr,PED,xyztrans,
    freq,counter_kE,zposition,R3,R,L,kpre,k)

% Extended Kalman filter measurement update

H=[cos(XE(10,1)),-sin(XE(10,1)),0,0,0,0,1,0,0,-XE(1,1)*sin(XE(10,1))-XE
    (2,1)*cos(XE(10,1)),0;
sin(XE(10,1)),cos(XE(10,1)),0,0,0,0,0,1,0,XE(1,1)*cos(XE(10,1))-XE(2,1)*
    sin(XE(10,1)),0;
0,0,1,0,0,0,0,0,1,0,0];

K=PP*H'/(H*PP*H'+R);
PE=(eye(L)-K*H)*PP*(eye(L)-K*H)' + K*R*K';
h=XP(7:9,1) + R3(XP(10,1))*XP(1:3,1);
XE=XP+K*(xyztrans(:,kpre)-h);

XE(11,1)=freq;

XED(:,end+1)=XE;
XEPD(:,end+1)=XE;
XEPDr(:,end+1)=[[XE(7:9,1)+R3(XE(10,1))*XE(1:3,1)];[R3(XE(10,1))*XE
    (4:6,1)];XE(7:11,1)];
XEDr(:,end+1)=[[XE(7:9,1)+R3(XE(10,1))*XE(1:3,1)];[R3(XE(10,1))*XE(4:6,1)
    ];XE(7:11,1)];
PED(:,end+1)=diag(PE);
zposition(:,end +1)=[XED(1,k+1);XED(3,k+1);100];
```

```
22 counter_kE(end+1)=counter_kE(end)+1;
```

SPF time update

```
1  function [XEu,XPu,PEu,PPu,X,XEPDu,XEPDur,PEuD,counter_k,counter_kE]=
       horizontal_spf_timeupdate...
2      (XEu,XPu,PEu,PPu,XEPDu,XEPDur,PEuD,Wc,Wm,counter_k,counter_kE,Q,R3,f,
           L,lambda,N,Qm)
3
4  % Sigma Point filter timeupdate
5
6  X=zeros(L,N);
7  if(det(PEu)<=0)
8  [U,S,V] = svd(PEu);
9  xsigma=sqrt(L+lambda)*U*sqrt(S)';
10 Xs=[XEu,XEu*ones(1,N-1)+[xsigma,-xsigma]];
11 else
12 xsigma=sqrt(L+lambda)*chol(PEu)';
13 Xs=[XEu,XEu*ones(1,N-1)+[xsigma,-xsigma]];
14 end
15 XPu=zeros(L,1);
16
17 for i=1:N
18     X(:,i)=f(Xs(:,i));
19     XPu=XPu+Wm(i)*X(:,i);
20 end
21
22 PPu=zeros(L,L);
23 for(i=1:N)
24     PPu=PPu+Wc(i)*((X(:,i)-XPu)*(X(:,i)-XPu)');
25 end
26 PPu=PPu+Qm;
27
28 XEu=XPu;
29 PEu=PPu;
30
31 XEPDu(:,end+1)=XEu;
32 XEPDur(:,end+1)=[[[XEu(7,1);XEu(8,1);XEu(9,1)]+R3(XEu(10))*XEu(1:3,1)];[
       R3(XEu(10))*XEu(4:6,1)];XEu(7:11,1)];
33 PEuD(:,end+1)=diag(PEu);
34 counter_k(end+1)=counter_k(end)+1;
35 counter_kE(end+1)=counter_kE(end)+1;
```

SPF measurement update

```
1  function [XEu,XPu,PEu,PPu,XEDu,XEPDu,XEDur,XEPDur,PEuD,X,counter_kE] =...
```

```
2                horizontal_spf_measupdate(XPu,XEu,PPu,PEu,XEDu,XEPDu,XEDur,
                     XEPDur,PEuD,Wc,Wm,xyztrans,freq,counter_kE,N,h,f,R3,R,m,L
                     ,lambda,kpre,X)
3
4  % Sigma Point filter measurement update
5
6  zp=zeros(m,1);
7  ZP=zeros(m,N);
8
9  for i=1:N
10       ZP(:,i)=h(X(:,i));
11       zp=zp+Wm(i)*ZP(:,i);
12 end
13
14 Pzz=(ZP-zp(:,ones(1,N)))*diag(Wc)*(ZP-zp(:,ones(1,N)))'+R;
15 Pxz=(X-XPu(:,ones(1,N)))*diag(Wc)*(ZP-zp(:,ones(1,N)))';
16 K=Pxz*inv(Pzz);
17 XEu=XPu+K*(xyztrans(:,kpre)-zp);
18 PEu=PPu-K*Pzz*K';
19
20 XEu(11,1)=freq;
21
22 XEPDu(:,end+1)=XEu;
23 XEDu(:,end+1)=XEu;
24 XEPDur(:,end+1)=[[[XEu(7,1);XEu(8,1);XEu(9,1)]+R3(XEu(10))*XEu(1:3,1)];[
       R3(XEu(10))*XEu(4:6,1)];XEu(7:11,1)];
25 XEDur(:,end+1)=[[[XEu(7,1);XEu(8,1);XEu(9,1)]+R3(XEu(10))*XEu(1:3,1)];[
       R3(XEu(10))*XEu(4:6,1)];XEu(7:11,1)];
26 PEuD(:,end+1)=diag(PEu);
27 counter_kE(end+1)=counter_kE(end);
```

Filters with non-linear process model, helix Motion

Main file

```
clear all
close all
format long;
T=0.01;                              % Discretization time

A=50;                                % Amplitude
w=2*pi*0.1;                          % Angular velocity
sT = 80;                             % Estimator on for 80 seconds
steps=10000;                         % Number of simulated steps
measurement_interval=1/T ;
initialization_phase = 2000;

number_of_monte_carlo_runs = 3;
dim=11;                              % Dimension of state vector
eXE_EKF=zeros(dim,measurement_interval*sT+sT-1,number_of_monte_carlo_runs
    );
eXE_SPF=zeros(dim,measurement_interval*sT+sT-1,number_of_monte_carlo_runs
    );
PE_EKF=zeros(dim,measurement_interval*sT+sT-1,number_of_monte_carlo_runs)
    ;
PE_SPF=zeros(dim,measurement_interval*sT+sT-1,number_of_monte_carlo_runs)
    ;

% Start Monte Carlo simulation
for(trajectory=1:number_of_monte_carlo_runs)

% Initialize helix motion
vtot=sqrt(170^2+(A*w)^2);
xa=zeros(7,steps);
xa(1,1)=0;
xa(2,1)=0;
xa(3,1)=50;
xa(4,1)=sqrt(vtot^2-(A*w)^2);
xa(5,1)=A*w;
xa(6,1)=0;
xa(7,1)=w;
course=zeros(3,steps);

xr=zeros(11,steps);
xr(1,1)=18000;
xr(2,1)=0;
xr(3,1)=50;

phi_init=pi+atan(xr(2,1)/xr(1,1));%+45*pi/180 % Heading angle
R_a_r=[cos(phi_init),-sin(phi_init),0;sin(phi_init),cos(phi_init)
    ,0;0,0,1]; % rotation matrix
```

```
42
43 % Measurement covariance in spherical coordinates
44 Rp=[30^2,0,0;0,0.001^2,0;0,0,0.001^2];
45
46 z(1:3,1:steps)=0;
47
48
49 % Simulate system in airplane frame
50 for t=1:steps-1
51     xa(1,t+1)=xa(1,t)+T/2*(xa(4,t)+xa(4,t));
52     xa(2,t+1)=xa(2,t)+T/2*(xa(5,t)+(xa(5,t)+T*(-w^2*xa(2,t))));
53     xa(3,t+1)=xa(3,t)+T/2*(xa(6,t)+(xa(6,t)+T*(-w^2*xa(3,t))));
54     xa(4,t+1)=xa(4,t);
55     xa(5,t+1)=xa(5,t)+T/2*(-w^2*xa(2,t)-w^2*(xa(2,t)+T*xa(5,t)));
56     xa(6,t+1)=xa(6,t)+T/2*(-w^2*xa(3,t)-w^2*(xa(3,t)+T*xa(6,t)));
57     xa(7,t+1)=xa(7,t);
58 end
59 % Simulate system in radar frame
60 p_ra_r=0; % Origo in airplane frame
61 for t=1:steps
62     xr(4:6,t)=R_a_r*xa(4:6,t);
63     xr(1:3,t)=xr(1:3,1)+R_a_r*xa(1:3,t);
64     if(t==initialization_phase)
65     p_ra_r=xr(1:3,t);
66     p_ra_r(3,1)=100;
67     end
68     xr(7:9,t)=p_ra_r;
69     xr(10,t)=phi_init;
70     xr(11,t)=w;
71
72     z(:,t)=[sqrt(xr(1,t)^2+xr(2,t)^2+xr(3,t)^2);atan(xr(2,t)/xr(1,t));acos
           ((xr(3,t))/(sqrt(xr(1,t)^2+xr(2,t)^2+xr(3,t)^2)))]+chol(Rp)*randn
           (3,1);
73 end
74
75 %-------------------------------------------------------------
76 % Extended Kalman Filter
77 %-------------------------------------------------------------
78
79 % Rotation matrix
80 R3 = @(x)[cos(x),-sin(x),0;sin(x),cos(x),0;0,0,1];
81
82 Q=eye(3)*0.05^2;
83 Ga=[zeros(3);eye(3);eye(3);[0,0,0;0,0,0]];
84 Qm=Ga*Q*Ga';
85
86 counter_k=0;
87 counter_kE=0;
88 counter_k(1)=1;
89 counter_kE(1)=1;
90 kpre=0;
91 k=0;
92 tk=0;
```

```
93
94  xyztransa=0;
95  xyztransa(1:3,1)=0;
96  xyztrans=0;
97  xyztrans(1:3,1)=0;
98  X_freq_est_ekf_pre=0;
99  X_freq_est_ekf_pre(1:3,1)=0;
100 c(1:3,1)=0;
101 course_xyz(1:3,1)=0;
102 XE(1:11,1)=0;
103 XED=XE;
104 XEPD=XE;
105 XEDr=XE;
106 XEPDr=XE;
107
108 for(t=1:steps-1)
109     if(mod(t,1000)==0)
110         tk=tk+1;
111         timeten(:,tk)=xr(:,t);
112     end
113
114     if t≤initialization_phase
115         if(mod(t,measurement_interval)==0)
116             kpre=kpre+1;
117             xyztransform = meastransf(z,Rp,t);
118             xyztrans(:,kpre)=xyztransform;
119
120             X_freq_est_ekf_pre(:,kpre)=xyztrans(:,kpre);
121             if(mod(kpre,10)==0)
122                 freq=music(X_freq_est_ekf_pre,T,measurement_interval);
123                 XE(11,1)=freq;
124             end
125
126             %Initializations
127             c(:,kpre)=xyztrans(:,kpre);
128             [course_x,course_y,course_z] = courseest(c,kpre);
129             course_xyz(1:3,kpre)=[course_x;course_y;course_z];
130
131             if(kpre==20)
132
133             if(course_xyz(1,end)>course_xyz(1,1))
134                 phi=3*pi/2 + atan((course_xyz(2,1)-course_xyz(2,end))/(
                        course_xyz(1,end)-course_xyz(1,1)));
135             else
136                 phi=pi+atan((course_xyz(2,1)-course_xyz(2,end))/(course_xyz
                        (1,1)-course_xyz(1,end)));
137             end
138             for i=1:kpre
139                 xyztransa(:,i)=R3(phi)'*xyztrans(:,i);
140             end
141
142             p0r=course_xyz(:,end);
143
```

```
144             XE(1:3,1)=[0;0;0];
145             %Initialize velocities
146             xvelocity=(xyztransa(1,end)-xyztransa(1,1))./length(xyztransa);
147             [yvelocitymin,indexmin]=min(xyztransa(2,:));
148             [yvelocitymax,indexmax]=max(xyztransa(2,:));
149             yvelocity=(abs(yvelocitymax-yvelocitymin))/(abs(indexmin-
                    indexmax));
150             [zvelocitymin,indexmin]=min(xyztransa(3,:));
151             [zvelocitymax,indexmax]=max(xyztransa(3,:));
152             zvelocity=(abs(zvelocitymax-zvelocitymin))/(abs(indexmin-
                    indexmax));
153             XE(4:6,1)=[xvelocity;yvelocity;zvelocity];
154
155             XE(7:9,1)=p0r;
156             XE(10,1)=phi;
157             XP=XE;
158
159             XED(1:length(XE),1)=XE;
160             XEPD(1:length(XE),1)=XE;
161
162             XEPDr(1:length(XE),1)=XE;
163             XEDr(1:1:length(XE),1)=XE;
164             XEPDr(:,1)=[[p0r+R3(phi)*XE(1:3,1)];[R3(phi)*XE(4:6,1)];XE
                    (7:11,1)];
165             XEDr(:,1)=[[p0r+R3(phi)*XE(1:3,1)];[R3(phi)*XE(4:6,1)];XE
                    (7:11,1)];
166
167             zposition=0;
168             zposition(1,1)=XED(1,1);
169             zposition(2,1)=XED(3,1);
170             zposition(3,1)=100;
171             L=numel(XE);
172
173             % Transform measurement covariance to cartesian coordinates
174             r=z(1,t);sr=sqrt(Rp(1,1));a=z(2,t);sa=sqrt(Rp(2,2));e=z(3,t);se
                    =sqrt(Rp(3,3));
175             R=sphertocartR(r,a,e,sr,sa,se);
176
177             PE=zeros(L);
178             PE(1:9,1:9)=[R^2,zeros(3),zeros(3);zeros(3),R,zeros(3);zeros(3)
                    ,zeros(3),R];
179             PE(10,10)=0.1;
180             PE(11,11)=0.1;
181             PP=PE;
182             PED=0;
183             PED(1:L,1)=diag(PE);
184             end
185         end
186     else %start estimator
187     if(mod(t,measurement_interval)==0)
188               k=k+1;
189               kpre=kpre+1;
190
```

```
191                 xyztransform = meastransf(z,Rp,t);
192                 xyztrans(:,kpre)=xyztransform;
193
194                 X_freq_est_ekf_pre(:,kpre)=xyztrans(:,kpre);
195                 freq=music(X_freq_est_ekf_pre,T,measurement_interval);
196                 XE(11,1)=freq;
197                 XP(11,1)=freq;
198
199                 %calculate measurement covariance
200                 r=z(1,t);sr=sqrt(Rp(1,1));a=z(2,t);sa=sqrt(Rp(2,2));e=z(3,t);
                        se=sqrt(Rp(3,3));
201                 R=sphertocartR(r,a,e,sr,sa,se);
202
203                 %Measurement update----------------------------
204                 [XE,XP,PE,PP,XED,XEPD,XEDr,XEPDr,PED,zposition,counter_kE] =
                        ...
205                     helix_ekf_measupdate(XE,XP,PP,XED,XEPD,XEDr,XEPDr,PED,
                            xyztrans,freq,counter_kE,zposition,R3,R,L,kpre,k);
206
207                 %Time update----------------------------------
208                 [XE,XP,PE,PP,XEPD,XEPDr,PED,counter_k,counter_kE]=...
209                     helix_ekf_timeupdate(XE,PE,XEPD,XEPDr,PED,counter_k,
                            counter_kE,R3,T,Qm);
210
211         else%Time update
212             [XE,XP,PE,PP,XEPD,XEPDr,PED,counter_k,counter_kE]=...
213                 helix_ekf_timeupdate(XE,PE,XEPD,XEPDr,PED,counter_k,
                        counter_kE,R3,T,Qm);
214         end
215
216         end
217 end %Extended kalman filter
218
219 %-------------------------------------------------------------
220 % Sigma Point Filter
221 %-------------------------------------------------------------
222 L=11;
223 N=2*L+1;
224 m = size(z,1);
225 Xa=zeros(L,N);
226 X=zeros(L,N);
227
228 alpha=0.001;
229 kappa=0;
230 beta=2;
231 lambda=alpha^2*(L+kappa)-L;
232
233 Wm=[lambda/(L+lambda)  0.5/(L+lambda)+zeros(1,2*L)];
234 Wc=Wm;
235 Wc(1)=Wc(1)+(1-alpha^2+beta);
236
237 R3 = @(x)[cos(x),-sin(x),0;sin(x),cos(x),0;0,0,1];
```

```
238 f = @(x)[x(1,1) + T*x(4,1); x(2,1) + T*x(5,1); x(3,1) + T*x(6,1); x(4,1);
        x(5,1) - T*x(11,1)^2*x(2,1); x(6,1) - T*x(11,1)^2*x(3,1); x(7,1);x
        (8,1);x(9,1);x(10,1);x(11,1)];
239 h = @(x)[[x(7,1);x(8,1);x(9,1)] + [cos(x(10,1)),-sin(x(10,1)),0;sin(x
        (10,1)),cos(x(10,1)),0;0,0,1]*[x(1,1);x(2,1);x(3,1)]];
240
241 counter_k=0;
242 counter_kE=0;
243 counter_k(1)=1;
244 counter_kE(1)=1;
245 kpre=0;
246 k=0;
247 tk=0;
248 tell=0;
249 timeup=0;
250
251 XEu(1:L,1)=0;
252 xyztransa=0;
253 xyztransa(1:3,1)=0;
254 xyztrans=0;
255 xyztrans(1:3,1)=0;
256 X_freq_est_spf_pre=0;
257 X_freq_est_spf_pre(1:3,1)=0;
258 c(1:3,1)=0;
259 course_xyz(1:3,1)=0;
260
261 XEDu=XEu;
262 XEPDu=XEu;
263 XEDur=XEu;
264 XEPDur=XEu;
265
266 for(t=1:steps-1)
267
268     if(mod(t,1000)==0)
269         tk=tk+1;
270         timeten(:,tk)=xr(:,t);
271     end
272
273     if t≤initialization_phase
274         if(mod(t,measurement_interval)==0)
275             kpre=kpre+1;
276             xyztransform = meastransf(z,Rp,t);
277             xyztrans(:,kpre)=xyztransform;
278
279             c(:,kpre)=xyztrans(:,kpre);
280             [course_x,course_y,course_z] = courseest(c,kpre);
281             course_xyz(1:3,kpre)=[course_x;course_y;course_z];
282
283             X_freq_est_spf_pre(:,kpre)=xyztrans(:,kpre);
284             if(mod(kpre,10)==0)
285                 freq=music(X_freq_est_spf_pre,T,measurement_interval);
286                 XEu(11,1)=freq;
287             end
```

```
288
289             if(kpre==20)
290
291             if(course_xyz(1,end)>course_xyz(1,1))
292                 phi=3*pi/2 + atan((course_xyz(2,1)-course_xyz(2,end))/(
                        course_xyz(1,end)-course_xyz(1,1)))
293             else
294                 phi=pi+atan((course_xyz(2,1)-course_xyz(2,end))/(course_xyz
                        (1,1)-course_xyz(1,end)))
295             end
296             for i=1:kpre
297             xyztransa(:,i)=R3(phi)'*xyztrans(:,i);
298             end
299             p0r=course_xyz(:,end);
300
301             XEu(1:3,1)=[0;0;0];
302
303             % Initialization of velocities
304             xvelocity=(xyztransa(1,end)-xyztransa(1,1))./length(xyztransa);
305             [yvelocitymin,indexmin]=min(xyztransa(2,:));
306             [yvelocitymax,indexmax]=max(xyztransa(2,:));
307             yvelocity=(abs(yvelocitymax-yvelocitymin))/(abs(indexmin-
                    indexmax));
308             [zvelocitymin,indexmin]=min(xyztransa(3,:));
309             [zvelocitymax,indexmax]=max(xyztransa(3,:));
310             zvelocity=(abs(zvelocitymax-zvelocitymin))/(abs(indexmin-
                    indexmax));
311             XEu(4:6,1)=[xvelocity;yvelocity;zvelocity];
312
313             XEu(7:9,1)=p0r;
314             XEu(10,1)=phi;
315             XPu=XEu;
316             if kpre==2
317                 XEu0=XEu;
318             end
319
320             XEDu(1:length(XEu),1)=XEu;
321             XEPDu(1:length(XEu),1)=XEu;
322
323             XEPDur(1:length(XEu),1)=XEu;
324             XEDur(1:length(XEu),1)=XEu;
325             XEPDur(:,1)=[[p0r+R3(phi)*XEu(1:3,1)];[R3(phi)*XEu(4:6,1)];XEu
                    (7:11,1)];
326             XEDur(:,1)=[[p0r+R3(phi)*XEu(1:3,1)];[R3(phi)*XEu(4:6,1)];XEu
                    (7:11,1)];
327
328             L=numel(XEu);
329
330             % Transform measurement covariance to cartesian coordinates
331             r=z(1,t);sr=sqrt(Rp(1,1));a=z(2,t);sa=sqrt(Rp(2,2));e=z(3,t);se
                    =sqrt(Rp(3,3));
332             R=sphertocartR(r,a,e,sr,sa,se);
333
```

```
334             PEu=zeros(L);
335             PEu(1:9,1:9)=[R^2,zeros(3),zeros(3);zeros(3),R,zeros(3);zeros
                    (3),zeros(3),R];
336             PEu(10,10)=0.1;
337             PEu(11,11)=0.1;
338             PPu=PEu;
339
340             PEuD=0;
341             PEuD(1:L,1)=diag(PEu);
342             end
343         end
344     else %start estimator
345         if(mod(t,measurement_interval)==0) %measurement update
346               k=k+1;
347               kpre=kpre+1;
348               xyztransform = meastransf(z,Rp,t);
349               xyztrans(:,kpre)=xyztransform;
350
351               r=z(1,t);sr=sqrt(Rp(1,1));a=z(2,t);sa=sqrt(Rp(2,2));e=z(3,t);
                      se=sqrt(Rp(3,3));
352              R=sphertocartR(r,a,e,sr,sa,se);
353
354               X_freq_est_spf_pre(:,kpre)=xyztrans(:,kpre);
355               freq=music(X_freq_est_spf_pre,T,measurement_interval);
356
357              %Measurement update
                      ------------------------------------------------------
358               [XEu,XPu,PEu,PPu,XEDu,XEPDu,XEDur,XEPDur,PEuD,X,counter_kE]
                      =...
359               helix_spf_measupdate(XPu,PPu,XEDu,XEPDu,XEDur,XEPDur,PEuD,Wc,
                      Wm,xyztrans,freq,counter_kE,N,h,R3,R,m,kpre,X);
360              %Time update
                      ------------------------------------------------------
361               [XEu,XPu,PEu,PPu,X,XEPDu,XEPDur,PEuD,counter_k,counter_kE]=
                      helix_spf_timeupdate(XEu,XPu,PEu,PPu,XEPDu,XEPDur,PEuD,Wc
                      ,Wm,counter_k,counter_kE,Qm,R3,f,L,lambda,N);
362          else %Time update
                 ------------------------------------------------------
363               [XEu,XPu,PEu,PPu,X,XEPDu,XEPDur,PEuD,counter_k,counter_kE]=
                      helix_spf_timeupdate(XEu,XPu,PEu,PPu,XEPDu,XEPDur,PEuD,Wc
                      ,Wm,counter_k,counter_kE,Qm,R3,f,L,lambda,N);
364         end
365     end
366 end
367
368 Time=counter_k.*T;
369 TimeE=counter_kE.*T;
370
371 %Calculate time for estimate
372 TimeEst=0;
373 for(t=1:10:length(Time))
374     if(t==1)
375         TimeEst(t)=T;
```

```
376     else
377         TimeEst(end+1)=Time(t-1);
378     end
379 end
380
381 %calculate xe time, new xe1.
382 prem=20;
383 xr1=zeros(dim,length(TimeE));
384 xrnew=xr(:,2000:end);
385 xr1(:,1)=xr(:,2000);
386 k=0;
387 for(t=1:length(TimeE))
388     k=k+1;
389     xr1(:,t)=xrnew(:,k);
390     if(t~=length(TimeE))
391     if(TimeE(t+1)==TimeE(t))
392         xr1(:,t+1)=xr1(:,t);
393         if(t~=length(TimeE))
394             t=t+1;
395             k=k-1;
396         end
397     end
398     end
399 end
400
401 %error trajectories
402 eXE_EKF(:,:,trajectory)=xr1-XEPDr;
403 eXE_SPF(:,:,trajectory)=xr1-XEPDur;
404 PE_EKF(:,:,trajectory)=PED;
405 PE_SPF(:,:,trajectory)=PEuD;
406 end %Monte Carlo Simulations
407
408 % Monte Carlo analysis
409 [mXE_EKF,mXE_SPF,S_EKF_true,S_SPF_true,S_EKF,S_SPF] = MCanalysis_airplane
        (eXE_EKF,eXE_SPF,PE_EKF,PE_SPF,dim,measurement_interval,
        number_of_monte_carlo_runs,sT);
```

EKF time update.

```
1 function [XE,XP,PE,PP,XEPD,XEPDr,PED,counter_k,counter_kE]=
      helix_ekf_timeupdate(XE,PE,XEPD,XEPDr,PED,counter_k,counter_kE,R3,T,
      Qm)
2
3 % Extended Kalman filter time update
4
5 XP(1,1)=XE(1)+T*XE(4);
6 XP(2,1)=XE(2)+T*XE(5);
7 XP(3,1)=XE(3)+T*XE(6);
8 XP(4,1)=XE(4);
9 XP(5,1)=XE(5)-XE(11)^2*XE(2)*T;
```

```
XP(6,1)=XE(6)-XE(11)^2*XE(3)*T;
XP(7,1)=XE(7);
XP(8,1)=XE(8);
XP(9,1)=XE(9);
XP(10,1)=XE(10);
XP(11,1)=XE(11);

F=[0,0,0,1,0,0,0,0,0,0,0;
    0,0,0,0,1,0,0,0,0,0,0;
    0,0,0,0,0,1,0,0,0,0,0;
    0,0,0,0,0,0,0,0,0,0,0;
    0,-XE(11,1)^2,0,0,0,0,0,0,0,0,-2*XE(11,1)*XE(2,1);
    0,0,-XE(11,1)^2,0,0,0,0,0,0,0,-2*XE(11,1)*XE(3,1);
    0,0,0,0,0,0,0,0,0,0,0;
    0,0,0,0,0,0,0,0,0,0,0;
    0,0,0,0,0,0,0,0,0,0,0;
    0,0,0,0,0,0,0,0,0,0,0;
    0,0,0,0,0,0,0,0,0,0,0];

Fi=expm(F*T);
PP=Fi*PE*Fi'+Qm;

XE=XP;
PE=PP;

PED(:,end+1)=diag(PE);
XEPD(:,end+1)=XE;
XEPDr(:,end+1)=[[XE(7:9,1)+R3(XE(10,1))*XE(1:3,1)];[R3(XE(10,1))*XE
    (4:6,1)];XE(7:11,1)];
counter_k(end+1)=counter_k(end)+1;
counter_kE(end+1)=counter_kE(end)+1;
```

EKF measurement update

```
function [XE,XP,PE,PP,XED,XEPD,XEDr,XEPDr,PED,zposition,counter_kE] =
    helix_ekf_measupdate(XE,XP,PP,XED,XEPD,XEDr,XEPDr,PED,xyztrans,freq,
    counter_kE,zposition,R3,R,L,kpre,k)

% Extended Kalman filter measurement update

H=[cos(XE(10,1)),-sin(XE(10,1)),0,0,0,0,0,1,0,0,-XE(1,1)*sin(XE(10,1))-XE
    (2,1)*cos(XE(10,1)),0;
sin(XE(10,1)),cos(XE(10,1)),0,0,0,0,0,0,1,0,XE(1,1)*cos(XE(10,1))-XE(2,1)*
    sin(XE(10,1)),0;
0,0,1,0,0,0,0,0,0,1,0,0];

K=PP*H'/(H*PP*H'+R);
PE=(eye(L)-K*H)*PP*(eye(L)-K*H)' + K*R*K';
h=XP(7:9,1) + R3(XP(10,1))*XP(1:3,1);
XE=XP+K*(xyztrans(:,kpre)-h);
```

```
13
14 XE(11,1)=freq;
15
16 XED(:,end+1)=XE;
17 XEPD(:,end+1)=XE;
18
19 PED(:,end+1)=diag(PE);
20 XEPDr(:,end+1)=[[XE(7:9,1)+R3(XE(10,1))*XE(1:3,1)];[R3(XE(10,1))*XE
       (4:6,1)];XE(7:11,1)];
21 XEDr(:,end+1)=[[XE(7:9,1)+R3(XE(10,1))*XE(1:3,1)];[R3(XE(10,1))*XE(4:6,1)
       ];XE(7:11,1)];
22  zposition(:,end +1)=[XED(1,k+1);XED(3,k+1);100];
23 counter_kE(end+1)=counter_kE(end)+1;
```

SPF time update

```
1  function [XEu,XPu,PEu,PPu,X,XEPDu,XEPDur,PEuD,counter_k,counter_kE]=
       helix_spf_timeupdate...
2       (XEu,XPu,PEu,PPu,XEPDu,XEPDur,PEuD,Wc,Wm,counter_k,counter_kE,Qm,R3,f
            ,L,lambda,N)
3
4  % Sigma Point filter time update
5
6  X=zeros(L,N);
7  xsigma=sqrt(L+lambda)*chol(PEu)';
8  Xs=[XEu,XEu*ones(1,N-1)+[xsigma,-xsigma]];
9  XPu=zeros(L,1);
10
11 for i=1:N
12      X(:,i)=f(Xs(:,i));
13      XPu=XPu+Wm(i)*X(:,i);
14 end
15
16 PPu=zeros(L,L);
17 for(i=1:N)
18      PPu=PPu+Wc(i)*((X(:,i)-XPu)*(X(:,i)-XPu)');
19 end
20 PPu=PPu+Qm;
21
22 XEu=XPu;
23 PEu=PPu;
24
25 PEuD(:,end+1)=diag(PEu);
26 XEPDu(:,end+1)=XEu;
27 XEPDur(:,end +1)=[[[XEu(7,1);XEu(8,1);XEu(9,1)]+R3(XEu(10))*XEu(1:3,1)];[
       R3(XEu(10))*XEu(4:6,1)];XEu(7:11,1)];
28 counter_k(end+1)=counter_k(end)+1;
29 counter_kE(end+1)=counter_kE(end)+1;
```

SPF measurement update

```
function [XEu,XPu,PEu,PPu,XEDu,XEPDu,XEDur,XEPDur,PEuD,X,counter_kE] =...
    helix_spf_measupdate(XPu,PPu,XEDu,XEPDu,XEDur,XEPDur,PEuD,Wc,Wm,
        xyztrans,freq,counter_kE,N,h,R3,R,m,kpre,X)

% Sigma Point filter measurement update

zp=zeros(m,1);
ZP=zeros(m,N);

for i=1:N
    ZP(:,i)=h(X(:,i));
    zp=zp+Wm(i)*ZP(:,i);
end

Pzz=(ZP-zp(:,ones(1,N)))*diag(Wc)*(ZP-zp(:,ones(1,N)))'+R;
Pxz=(X-XPu(:,ones(1,N)))*diag(Wc)*(ZP-zp(:,ones(1,N)))';
K=Pxz*inv(Pzz);
XEu=XPu+K*(xyztrans(:,kpre)-zp);
PEu=PPu-K*Pzz*K';

XEu(11,1)=freq;

PEuD(:,end+1)=diag(PEu);
XEPDu(:,end+1)=XEu;
XEDu(:,end+1)=XEu;
XEPDur(:,end+1)=[[[XEu(7,1);XEu(8,1);XEu(9,1)]+R3(XEu(10))*XEu(1:3,1)];[
    R3(XEu(10))*XEu(4:6,1)];XEu(7:11,1)];
XEDur(:,end+1)=[[[XEu(7,1);XEu(8,1);XEu(9,1)]+R3(XEu(10))*XEu(1:3,1)];[
    R3(XEu(10))*XEu(4:6,1)];XEu(7:11,1)];
counter_kE(end+1)=counter_kE(end);
```

MUSIC

```
function [freq] = music(X_freq_est_spf_pre,T,measurement_interval)
% MUSIC algorithm
len=length(X_freq_est_spf_pre);
if(len<11)
    x_music = X_freq_est_spf_pre(1,1:length(X_freq_est_spf_pre)-1);
    y_music = X_freq_est_spf_pre(2,1:length(X_freq_est_spf_pre)-1);
end

if (len>20)
    x_music = X_freq_est_spf_pre(1,len-20+1:len);
    y_music = X_freq_est_spf_pre(2,len-20+1:len);
end
if (len>30)
    x_music = X_freq_est_spf_pre(1,len-30+1:len);
    y_music = X_freq_est_spf_pre(2,len-30+1:len);
end
if (len>40)
    x_music = X_freq_est_spf_pre(1,len-40+1:len);
    y_music = X_freq_est_spf_pre(2,len-40+1:len);
end
if (len>50)
    x_music = X_freq_est_spf_pre(1,len-50+1:len);
    y_music = X_freq_est_spf_pre(2,len-50+1:len);
end

poly=polyfit(x_music,y_music,1);
regression=poly(1)*x_music+poly(2);
y_music=y_music-regression;
m=length(y_music)/2.5;
poles=2;
T=T*measurement_interval;

%Hankel Data matrix:
y_music=y_music';
[row,col] = size(y_music);

nn = row+1-m;
X = [];
for i=1:m
    X=[X ; y_music(i:i+nn-1,1)'];
end

%Signal subspace matrix:
[U,S,V] = svd(X,0);
es = U(:,1:poles);
[mm,nn] = size(es);

fgrid = (-0.05:0.001:0.5);
fgrid = fgrid/(T);
```

```
50
51 ster = 2*pi*(fgrid(:))'*T;
52 k = (0:m-1)';
53 temp = exp(sqrt(-1)*k*ster);
54 if nn == 1
55     p1 = abs(es'*temp).^2;
56     P = ones(1,length(p1))./(m-p1);
57 else
58     p1 = sum(abs(es'*temp).^2);
59     p = ones(1,length(p1))./(m-p1);
60 end
61 p = p/max(p);
62
63 [maximum,index]=max(p);
64 freq=2*pi*fgrid(index);
```

Measurement and Covariance Transformations

```
function xyztransform = meastransf(z,Rp,t)

% Transforme measurements from spherical to cartesian coordinates

xyztransform=[exp(Rp(2,2)/2)*exp(Rp(3,3)/2)*z(1,t)*sin(z(3,t))*cos(z(2,t));
 exp(Rp(2,2)/2)*exp(Rp(3,3)/2)*z(1,t)*sin(z(3,t))*sin(z(2,t));
 exp(Rp(3,3)/2)*z(1,t)*cos(z(3,t))];
```

```
function R=sphertocartR(r,a,e,sr,sa,se)

% Transforme measurement covariance from spherical to cartesian
    coordinates

lambdae = exp(-se^2/2);
lambdaep = exp(-2*se^2);
lambdaa = exp(-sa^2/2);
lambdaap = exp(-2*sa^2);
a = a*pi/180;
e = e*pi/180;

R(1,1)=1/4*lambdaa^(-2)*lambdae^(-2)*(r^2+2*sr^2)*(1+lambdaap^2*cos(2*a))
    *(1+lambdaep^2*cos(2*e))-1/4*(r^2+sr^2)*(1+lambdaap*cos(2*a))*(1+
    lambdaep*cos(2*e));

R(2,2)=1/4*lambdaa^(-2)*lambdae^(-2)*(r^2+2*sr^2)*(1-lambdaap^2*cos(2*a))
    *(1+lambdaep^2*cos(2*e))-1/4*(r^2+sr^2)*(1-lambdaap*cos(2*a))*(1+
    lambdaep*cos(2*e));

R(3,3)=1/2* lambdae^(-2)*(r^2+2*sr^2)*(1-lambdaep^2*cos(2*e))-1/2*(r^2+sr
    ^2)* (1-lambdaep*cos(2*e));

R(1,2)=1/4*lambdaa^(-2)*lambdae^(-2)*lambdaap^2*(r^2+2*sr^2)* sin(2*a)
    *(1+lambdaep^2*cos(2*e))-1/4*(r^2+sr^2)* lambdaap*sin(2*a) *(1+
    lambdaep*cos(2*e));

R(1,3)=1/2*lambdaa *lambdae^(-2)*(r^2+2*sr^2)* cos(a)*lambdaep^2*sin(2*e)
    -1/2*(r^2+sr^2)* lambdaa *cos(a) * lambdaep*sin(2*e);

R(2,3)=1/2*lambdaa *lambdae^(-2)*(r^2+2*sr^2)* sin(a)*lambdaep^2*sin(2*e)
    -1/2*(r^2+sr^2)* lambdaa *sin(a) * lambdaep*sin(2*e);

R(2,1)=R(1,2);
R(3,1)=R(1,3);
R(3,2)=R(2,3);
```

Course Estimate

```
function [course_x,course_y,course_z] = courseest(c,k)

% Course estimate in initialization phase of the filters
% Polyfit() calculates linear regression from the measurements
% given up to present time.

poly=polyfit(c(1,1:k),c(2,1:k),1);
course_xy=poly(1)*c(1,1:k)+poly(2);

poly=polyfit(course_xy(1,1:k),c(3,1:k),1);
course_xy_z=poly(1)*course_xy(1,1:k)+poly(2);

course_x=c(1,k);
course_y=course_xy(k);
course_z=course_xy_z(k);
```

Monte Carlo Analysis

```
function [mXE_EKF,mXE_SPF,S_EKF_true,S_SPF_true,S_EKF,S_SPF] =
    MCanalysis_airplane(eXE_EKF,eXE_SPF,PE_EKF,PE_SPF,dim,
    measurement_interval,number_of_monte_carlo_runs,sT)

% Monte Carlo Analysis

%means
mXE_EKF=zeros(dim,measurement_interval*sT+sT-1);
mXE_SPF=zeros(dim,measurement_interval*sT+sT-1);

for(trajectory=1:number_of_monte_carlo_runs)
   mXE_EKF=mXE_EKF+eXE_EKF(:,:,trajectory);
   mXE_SPF=mXE_SPF+eXE_SPF(:,:,trajectory);
end

mXE_EKF=mXE_EKF/number_of_monte_carlo_runs;
mXE_SPF=mXE_SPF/number_of_monte_carlo_runs;

%Calculate covariances and standard deviations
PE_EKF_true=zeros(dim,dim,measurement_interval*sT+sT-1);
PE_SPF_true=zeros(dim,dim,measurement_interval*sT+sT-1);

for(trajectory=1:number_of_monte_carlo_runs)
    for(k=1:measurement_interval*sT+sT-1)
        PE_EKF_true(:,:,k)=PE_EKF_true(:,:,k)+(eXE_EKF(:,k,trajectory)-
            mXE_EKF(:,k))*(eXE_EKF(:,k,trajectory)-mXE_EKF(:,k))';
```

```
24              PE_SPF_true(:,:,k)=PE_SPF_true(:,:,k)+(eXE_SPF(:,k,trajectory)-
                    mXE_SPF(:,k))*(eXE_SPF(:,k,trajectory)-mXE_SPF(:,k))';
25          end
26      end
27
28      PE_EKF=sum(PE_EKF,3)/(number_of_monte_carlo_runs-1);
29      PE_SPF=sum(PE_SPF,3)/(number_of_monte_carlo_runs-1);
30
31      PE_EKF_true=PE_EKF_true/(number_of_monte_carlo_runs-1);
32      PE_SPF_true=PE_SPF_true/(number_of_monte_carlo_runs-1);
33
34      S_EKF=zeros(dim,measurement_interval*sT+sT-1);
35      S_SPF=zeros(dim,measurement_interval*sT+sT-1);
36      S_EKF_true=zeros(dim,measurement_interval*sT+sT-1);
37      S_SPF_true=zeros(dim,measurement_interval*sT+sT-1);
38
39      for(k=1:measurement_interval*sT+sT-1)
40          S_EKF(:,k)=sqrt(PE_EKF(:,k));
41          S_SPF(:,k)=sqrt(PE_SPF(:,k));
42          S_EKF_true(:,k)=sqrt(diag(PE_EKF_true(:,:,k)));
43          S_SPF_true(:,k)=sqrt(diag(PE_SPF_true(:,:,k)));
44      end
```

Falling Body

Main file

```
%fallingbody
%Radar tracking of a falling body
clear all
close all

sT = 30;                    % simulation time
T=0.1;                      % Discretization time
measurement_interval=1/T;
steps=300;                  % 300 time steps
number_of_monte_carlo_runs = 5;
dim=3;                      % State vector dimension

% Error trajectories
eXE_EKF=zeros(dim,measurement_interval*sT+sT-1,number_of_monte_carlo_runs
    );
eXE_SPF=zeros(dim,measurement_interval*sT+sT-1,number_of_monte_carlo_runs
    );
PE_EKF=zeros(dim,dim,measurement_interval*sT+sT-1,
    number_of_monte_carlo_runs);
PE_SPF=zeros(dim,dim,measurement_interval*sT+sT-1,
    number_of_monte_carlo_runs);
% Start Monte Carlo Simulation
for(trajectory=1:number_of_monte_carlo_runs)
% Initializations
g=9.81;                     % Gravity
gamma=1.754;
eta=1.49*10^-4;
beta=19161;                 % Ballistic coefficient.

% Initializations of filter
r=200^2;
P=[1000000, 0, 0; 0,1000000, 0; 0, 0, 10000000];
x=[60000+sqrt(r)*randn;4000;18000];

% True trajectory initialization
xr(:,1)= [61000; 3048; 19161];

% Simulation of true trajectory
for i = 1:steps-1
    xr(1,i+1) = xr(1,i) - T*xr(2,i);
    xr(2,i+1) = xr(2,i) -((gamma*exp(-eta*xr(1,i))*g*xr(2,i)^2)*T)/(2*
        beta) +g*T;
    xr(3,i+1) = beta;
end

% Defining measurements with measurement noise
```

```
for i = 1:steps-1
    z(i) = xr(1,i) + sqrt(r)*randn;
end
%-----------------------------------------------------
% Extended Kalman Filter
%-----------------------------------------------------

%Initializations
XE=x;                       % x estimate
PE=P;                       % P estimate
PED=0;                      % PE Display for plot
PED(1:3,1:3)=PE;
XEPD=XE;                    % Matrix of stored estimates
H=[1,0,0];                  % Measurement matrix
La=[0;g;0];                 % Control matrix
u=1;
z_D(1)=xr(1,1);             % Stored measurements
counter_k=0;                % Counter for estimate and prediction
counter_kE=0;               % Counter for estimate
counter_k(1)=1;
counter_kE(1)=1;
Time=0;                     % Time in seconds for estimate and prediction
TimeE=0;                    % Time in secongs for estimate
Time(1)=1;
TimeE(1)=1;
k=0;

% START ESTIMATOR
% Time update 10Hz
% Measurement update 1 Hz
for(t=1:steps-1)
        if(mod(t,measurement_interval)==0)
            k=k+1;
            %Measurement update----------------------------
            zD(1,k)=t*T;
            zD(2,k)=z(:,t);
            [XE,PE,XEPD,PED,counter_kE] = fallingbody_EKF_measupdate(XP,
                PP,H,r,z,XEPD,PED,counter_kE,t);
            %Time update----------------------------------
            [XP,XE,PP,PE,PED,XEPD,counter_k,counter_kE] =
                fallingbody_EKF_timeupdate(XE,PE,XEPD,PED,counter_k,
                counter_kE,T);

        else % Time update
            [XP,XE,PP,PE,PED,XEPD,counter_k,counter_kE] =
                fallingbody_EKF_timeupdate(XE,PE,XEPD,PED,counter_k,
                counter_kE,T);
        end
end

Time=counter_k.*T;
TimeE=counter_kE.*T;

```

```
90
91 %-------------------------------------------------------------------
92 % Sigma Point Filter
93 %-------------------------------------------------------------------
94 % Initializations
95 L=dim;
96 N=2*L+1;                    % Number of sigma points
97 m = size(z,1);              % Dimension of measurement vector
98 Xa=zeros(L,N);              % Sigma point matrix initialization
99 X=zeros(L,N);               % Sigma point matrix initialization
100 XEu=x;
101 PEu=P;
102 PEuD=0;
103 PEuD(1:3,1:3)=PEu;
104 XEPDu=XEu;
105 z_Du(1)=xr(1,1);
106
107 % Sigma point filter tuning factors
108 alpha=0.001;
109 kappa=0;
110 beta=2;
111 lambda=alpha^2*(L+kappa)-L;
112
113 Wm=[lambda/(L+lambda)  0.5/(L+lambda)+zeros(1,2*L)];
114 Wc=Wm;
115 Wc(1)=Wc(1)+(1-alpha^2+beta);
116
117 counter_k=0;
118 counter_kE=0;
119 counter_k(1)=1;
120 counter_kE(1)=1;
121
122 f = @(x)[x(1,1) - T*x(2,1);
123     x(2,1)-((gamma*exp(-eta*x(1,1))*g*x(2,1)^(2))/(2*x(3,1)))*T + g*T;
124     x(3,1)];                % Non-linear state equations
125 h = @(x) x(1);              % Linear measurment equation
126
127 % START ESTIMATOR
128 % Time update 10Hz
129 % Measurement update 1 Hz
130 for(t=1:steps-1)
131     if(mod(t,measurement_interval)==0)
132         %Measurement update----------------------------------------
133         [XEu,PEu,XEPDu,PEuD,counter_kE] = fallingbody_SPF_measupdate(XPu,
                PPu,XEPDu,PEuD,H,r,z,counter_kE,dim,t);
134         %Time update----------------------------------------------
135         [XPu,XEu,PPu,PEu,XEPDu,PEuD,counter_k,counter_kE,X] =
                fallingbody_SPF_timeupdate(XEu,PEu,XEPDu,PEuD,counter_k,
                counter_kE,f,Wm,Wc,L,N,lambda);
136     else %Time update-------------------------------------------
137         [XPu,XEu,PPu,PEu,XEPDu,PEuD,counter_k,counter_kE,X] =
                fallingbody_SPF_timeupdate(XEu,PEu,XEPDu,PEuD,counter_k,
                counter_kE,f,Wm,Wc,L,N,lambda);
```

```
138      end
139  end
140
141  %Calculate time for estimate
142  TimeEst=0;
143  for(t=1:10:length(Time))
144      if(t==1)
145          TimeEst(t)=T;
146      else
147          TimeEst(end+1)=Time(t-1);
148      end
149  end
150
151  %calculate time for x true
152  %calculate xr time, new xr1.
153  xr1=zeros(dim,length(TimeE));
154  xr1(:,1)=xr(:,1);
155  k=0;
156  for(t=1:length(TimeE))
157      k=k+1;
158      xr1(:,t)=xr(:,k);
159      if(t≠length(TimeE))
160      if(TimeE(t+1)==TimeE(t))
161          xr1(:,t+1)=xr1(:,t);
162          if(t≠length(TimeE))
163              t=t+1;
164              k=k-1;
165          end
166      end
167      end
168  end
169
170  %error trajectories
171  eXE_EKF(:,:,trajectory)=xr1-XEPD;
172  eXE_SPF(:,:,trajectory)=xr1-XEPDu;
173  PE_EKF(:,:,:,trajectory)=PED;
174  PE_SPF(:,:,:,trajectory)=PEuD;
175  end %Monte Carlo Simulations
176
177  % Monte Carlo Analysis
178  [mXE_EKF,mXE_SPF,S_EKF_true,S_SPF_true,S_EKF,S_SPF] = MCanalysis(eXE_EKF,
         eXE_SPF,PE_EKF,PE_SPF,dim,measurement_interval,
         number_of_monte_carlo_runs,sT);
```

EKF time update.

```
1  function [XP,XE,PP,PE,PED,XEPD,counter_k,counter_kE] =
       fallingbody_EKF_timeupdate(XE,PE,XEPD,PED,counter_k,counter_kE,T)
2
3  %Extended Kalman filter time update
```

```
4
5  g=9.81;
6  gamma=1.754;
7  eta=1.49*10^-4;
8
9  XP(1,1)=XE(1,1)-T*XE(2,1);
10 XP(2,1)=XE(2,1)-gamma*exp(-eta*XE(1,1))*g*XE(2,1)^2*T/(2*XE(3,1))+g*T;
11 XP(3,1)=XE(3,1);
12
13 F=[0,-1,0;
14    (gamma*eta*exp(-eta*XE(1))*g*XE(2)^2)/(2*XE(3)),-(gamma*exp(-eta*XE(1))
          *g*XE(2))/(XE(3)),(gamma*exp(-eta*XE(1))*g*XE(2)^2)/(2*XE(3)^2);
15    0,0,0];
16
17 Fi=expm(F*T);
18 PP=Fi*PE*Fi';
19
20 XE=XP;
21 PE=PP;
22
23 XEPD(:,end+1)=XE;
24 PED(:,:,end+1)=PE;
25 counter_kE(end+1)=counter_kE(end)+1;
26 counter_k(end+1)=counter_k(end)+1;
```

EKF measurement update

```
1  function [XE,PE,XEPD,PED,counter_kE] = fallingbody_EKF_measupdate(XP,PP,H
       ,r,z,XEPD,PED,counter_kE,t)
2
3  % Extended Kalman filter measurement update.
4
5  K=PP*H'*inv(H*PP*H'+r);
6  PE=(eye(3)-K*H)*PP;
7  XE=XP+K*(z(t)-H*XP);
8
9  XEPD(:,end+1)=XE;
10 PED(:,:,end+1)=PE;
11 counter_kE(end+1)=counter_kE(end);
```

SPF time update

```
1  function [XPu,XEu,PPu,PEu,XEPDu,PEuD,counter_k,counter_kE,X] =
       fallingbody_SPF_timeupdate(XEu,PEu,XEPDu,PEuD,counter_k,counter_kE,f,
       Wm,Wc,L,N,lambda)
2
3  % Sigma Point filter timeupdate
```

```
4
5  X=zeros(L,N);
6  xsigma=sqrt(L+lambda)*chol(PEu)';
7  Xs=[XEu,XEu*ones(1,N-1)+[xsigma,-xsigma]];
8
9  XPu=zeros(L,1);
10 for  i=1:N
11       X(:,i)=f(Xs(:,i));
12       XPu=XPu+Wm(i)*X(:,i);
13 end
14
15 PPu=zeros(L,L);
16 for(i=1:N)
17       PPu=PPu+Wc(i)*((X(:,i)-XPu)*(X(:,i)-XPu)');
18 end
19
20 XEu=XPu;
21 PEu=PPu;
22
23 XEPDu(:,end+1)=XEu;
24 PEuD(:,:,end+1)=PEu;
25 counter_kE(end+1)=counter_kE(end)+1;
26 counter_k(end+1)=counter_k(end)+1;
```

SPF measurement update

```
1  function [XEu,PEu,XEPDu,PEuD,counter_kE] = fallingbody_SPF_measupdate(XPu
       ,PPu,XEPDu,PEuD,H,r,z,counter_kE,dim,t)
2
3  %Sigma Point filter measurement update
4
5  Ku=PPu*H'*inv(H*PPu*H'+r);
6  PEu=(eye(dim)-Ku*H)*PPu;
7  XEu=XPu+Ku*(z(t)-H*XPu);
8
9  XEPDu(:,end+1)=XEu;
10 PEuD(:,:,end+1)=PEu;
11 counter_kE(end+1)=counter_kE(end);
```

Monte Carlo Analysis

```
1  function [mXE_EKF,mXE_SPF,S_EKF_true,S_SPF_true,S_EKF,S_SPF] = MCanalysis
       (eXE_EKF,eXE_SPF,PE_EKF,PE_SPF,dim,measurement_interval,
       number_of_monte_carlo_runs,sT)
2
3  %means
4  mXE_EKF=zeros(dim,measurement_interval*sT+sT-1);
```

```
mXE_SPF=zeros(dim,measurement_interval*sT+sT-1);

for(trajectory=1:number_of_monte_carlo_runs)
    mXE_EKF=mXE_EKF+eXE_EKF(:,:,trajectory);
    mXE_SPF=mXE_SPF+eXE_SPF(:,:,trajectory);
end

mXE_EKF=mXE_EKF/number_of_monte_carlo_runs;
mXE_SPF=mXE_SPF/number_of_monte_carlo_runs;

%Calculate covariances and standard deviations
PE_EKF_true=zeros(dim,dim,measurement_interval*sT+sT-1);
PE_SPF_true=zeros(dim,dim,measurement_interval*sT+sT-1);

for(trajectory=1:number_of_monte_carlo_runs)
    for(k=1:measurement_interval*sT+sT-1)
        PE_EKF_true(:,:,k)=PE_EKF_true(:,:,k)+(eXE_EKF(:,k,trajectory)-
            mXE_EKF(:,k))*(eXE_EKF(:,k,trajectory)-mXE_EKF(:,k))';
        PE_SPF_true(:,:,k)=PE_SPF_true(:,:,k)+(eXE_SPF(:,k,trajectory)-
            mXE_SPF(:,k))*(eXE_SPF(:,k,trajectory)-mXE_SPF(:,k))';
    end

end

PE_EKF=sum(PE_EKF,4)/(number_of_monte_carlo_runs-1);
PE_SPF=sum(PE_SPF,4)/(number_of_monte_carlo_runs-1);

PE_EKF_true=PE_EKF_true/(number_of_monte_carlo_runs-1);
PE_SPF_true=PE_SPF_true/(number_of_monte_carlo_runs-1);

S_EKF=zeros(dim,measurement_interval*sT+sT-1);
S_SPF=zeros(dim,measurement_interval*sT+sT-1);
S_EKF_true=zeros(dim,measurement_interval*sT+sT-1);
S_SPF_true=zeros(dim,measurement_interval*sT+sT-1);

for(k=1:measurement_interval*sT+sT-1)
    S_EKF(:,k)=sqrt(diag(PE_EKF(:,:,k)));
    S_SPF(:,k)=sqrt(diag(PE_SPF(:,:,k)));
    S_EKF_true(:,k)=sqrt(diag(PE_EKF_true(:,:,k)));
    S_SPF_true(:,k)=sqrt(diag(PE_SPF_true(:,:,k)));
end
```

Bibliography

[1] Arthur Gelb, editor. *Applied Optimal Estimation*. The M.I.T. Press, 1974.

[2] Branko Ristic and Sanjeev Arulampalam and Neil Gordon. *Beyond The Kalman Filter*. Artech House Publishers, 2004.

[3] Chi-Tsong Chen. *Linear System Theory and Design*. Oxford University Press, 1999.

[4] Dan Simon. *Optimal State Estimation*. John Wiley and Sons, INC, 2006.

[5] Dirk Tenne and Tarunraj Singh. Circular Filters for Target Tracking in 3D. Technical report, 2004.
http://www.fusion2004.foi.se/papers/IF04-0370.pdf(2010-04-09).

[6] Don H. Johnson and Dan E. Dudgeon. *Array Signal Processing. Concepts and Techniques*. P T R Prentice-Hall, Inc, 1993.

[7] Eli Brookner. *Tracking and Kalman Filtering Made Easy*. John Wiley and Sons, INC, 1998.

[8] Greg Welch and Gary Bishop. An Introduction to the Kalman Filter. Technical Report 95–041, Department of Computer Science. University of North Carolina at Chapel Hill, 2003.

[9] Oddvar Hallingstad. Matematisk modellering av dynamiske systemer. Hand out from Oddvar Hallingstad during the course UNIK4540.

[10] Oddvar Hallingstad. Monte carlo- og kovariansanalyse av kalmanfilteret. Hand out from Oddvar Hallingstad during the course UNIK4500.

[11] S.J. Julier and J.K. Uhlmann. Unscented filtering and nonlinear estimation. *Proceedings of the IEEE*, 92(3):401 – 422, mar 2004.

[12] J.R. Katkuri, V.P. Jilkov, and X.R. Li. A comparative study of nonlinear filters for target tracking in mixed coordinates. In *System Theory (SSST), 2010 42nd Southeastern Symposium on*, pages 202 –207, 7-9 2010.

[13] Mo Longbin, Song Xiaoquan, Zhou Yiyu, Sun Zhong Kang, and Y. Bar-Shalom. Unbiased converted measurements for tracking. *Aerospace and Electronic Systems, IEEE Transactions on*, 34(3):1023 –1027, jul 1998.

[14] Olav Egeland and Jan Tommy Gravdahl. *Modelling and Simulation for Automatic Control*. Tapir Trykkeri, 2002.

[15] R.E. Kalman. A new approach to linear filtering and prediction problems. *Trans. ASME, Journal of Basic Engineering*, 82:35–45, March 1960.

[16] Ronald E. Walpole, Raymond H. Myers, Sharon L. Myers and Keying Ye. *Probability & Statistics for Engineers & Scientists, Eighth edition*. Pearson Prentice Hall, 2007.

[17] B. Sadeghi and B. Moshiri. Second-order ekf and unscented kalman filter fusion for tracking maneuvering targets. In *Information Reuse and Integration, 2007. IRI 2007. IEEE International Conference on*, pages 514 –519, 13-15 2007.

[18] Saeed V.Vaseghi. *Advanced Digital Signal Processing and Noise Reduction. Fourth Edition.* John Wiley and Sons, Ltd, Publications, 2008.

[19] L. Schwartz and E. Stear. A computational comparison of several nonlinear filters. *Automatic Control, IEEE Transactions on*, 13(1):83 – 86, feb 1968.

[20] Simon J. Julier and Jeffrey K. Uhlmann. A New Extension of the Kalman Filter to Nonlinear Systems. Technical report, 1997.
http://www.cs.unc.edu/~welch/kalman/media/pdf/Julier1997_SPIE_KF.pdf(2010-03-24).

[21] Eric E. Wan and Rudolph van der Merwe. The uncented kalman filter for nonlinear estimation. In *IEEE 2000 Adaptive Systems for Signal Processing, Communications, and Control Symposium*. IEEE, 2000.

[22] Yaakov Bar-Shalom, X. Rong Li, Thiagalingam Kirubarajan. *Estimation with Applications to Tracking and Navigation*. John Wiley and Sons, INC, 2001.